TECHNICAL REPORT ITL-92-10

(2)

AN IMAGE PROCESSING TECHNIQUE FOR ACHIEVING LOSSY COMPRESSION OF DATA AT RATIOS IN EXCESS OF 100:1

by

Michael G. Ellis

Information Technology Laboratory

DEPARTMENT OF THE ARMY
Waterways Experiment Station, Corps of Engineers
3909 Halls Ferry Road, Vicksburg, Mississippi 39180-6199

November 1992
Final Report

Prepared for DEPARTMENT OF THE ARMY
US Army Corps of Engineers
Washington, DC 20314-1000

Destroy this report when no longer needed. Do not return it to the originator.

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

The contents of this report are not to be used for advertising, publication, or promotional purposes. Citation of trade names does not constitute an official endorsement or approval of the use of such commercial products.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	November 1992	Final report

4. TITLE AND SUBTITLE

An Image Processing Technique for Achieving Lossy Compression of Data at Ratios in Excess of 100:1

5. FUNDING NUMBERS

6. AUTHOR(S)

Michael G. Ellis

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

US Army Engineer Waterways Experiment Station,
Information Technology Laboratory,
3909 Halls Ferry Road, Vicksburg, MS 39180-6199

8. PERFORMING ORGANIZATION REPORT NUMBER

Technical Report ITL-92-10

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

US Army Corps of Engineers
Washington, DC 20314-1000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

Available from National Technical Information Service, 5285 Port Royal Road, Springfield, VA 22161.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report presents a new method for image compression that achieves ratios in excess of 100:1. It begins as a tutorial of basic compression techniques that includes both lossless and lossy methods. The term "lossless" means that the compressed file can be reconstructed exactly, while "lossy" means that there is some loss of information in the reconstructed image. The most important components of the Joint Photographic Experts Group (JPEG) algorithm are targeted including Huffman encoding and the Cosine Transform. LZW, singular value decompostion, fractal compression, and other techniques are also discussed. The feasibility of using these methods for bandwidth reduction is analyzed relative to several High Definition TV (HDTV) standards. Finally, the cosine transform is modified to show how scanned images can be compressed 100:1 with little loss in quality. Several examples are presented.

14. SUBJECT TERMS		15. NUMBER OF PAGES
Compression	Lossy	217
Image processing	Video	16. PRICE CODE
Lossless		

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED		

Contents

Preface

This "Image Processing Technique for Achieving Lossy Compression of Data at Ratios in Excess of 100:1" documents a study by Mr. Michael G. Ellis, Electrical Engineer, Computer Science Division (CSD), Information Technology Laboratory (ITL), Waterways Experiment Station (WES), Vicksburg, Mississippi.

This report that describes the development of a new method for achieving compression ratios in excess of 100:1 begins by reviewing the standard lossless and lossy techniques that comprise existing algorithms. The term "lossless" means that the compressed file can be reconstructed to match the original, while "lossy" means that there is some loss of information in the reconstructed image. The utility of these methods to bandwidth reduction is analyzed relative to several High Definition TV (HDTV) standards.

Mr. Robert Moorhead, Professor of Electrical Engineering, Mississippi State University, contributed significantly to the success of the undertaking and provided many helpful suggestions. Mr. Roy Campbell, ITL, programmed the "Image Lab" software that was used in the research and development of these methods; Mr. Joel McAlister, ITL, also programmed many stand-alone utility programs and generated the hardcopy images shown in this document.

The work was accomplished at WES under the supervision of Dr. N. Radhakrishnan, Director, ITL, and Dr. Windell F. Ingram, Chief, CSD.

At the time of publication of this report, the Director of WES was Dr. Robert W. Whalin. The Commander was COL Leonard G. Hassell, EN.

1 Lossless Coding Techniques

Entropy Coding

The term "entropy" is borrowed from thermodynamics and has a similar meaning. The higher the entropy of a message, the more information it contains. The entropy of a data file is defined as the information per byte for that particular file. If the 8-bit ASCII character set that makes up a data or image file is not uniformly distributed, then the entropy will be less than 8 bits/byte. In entropy coding, the goal is to encode a block of M pixels each containing B bits, for a total of $M*B$ bits. The entropy for any general file is given as

$$H = -\sum_{i=1}^{M} [p_i \log_2 (p_i)] \qquad (1)$$

In an 8-bit character set, there are 256 values that can be represented by a single byte. The probability, p_i, is the total occurrences of the i_{th} byte divided by the total number of bytes in the data file.

The program, ENTROPY.BAS, uses Equation 1 to estimate the amount of compression achievable by entropy coding. ENTROPY.BAS works only under QuickBasic 4.5 because of the "open binary" statement in line 70. The C language version, ENTROPY.C, is provided to illustrate the syntax differences between BASIC and C. Most compression programs are written in C since the more optimized C compiler produces executables which are 10 times faster than compiled BASIC.

Huffman Compression

This classic compression technique, introduced in 1952 by David A. Huffman, is perhaps the most well-known for entropy compression. It

achieves the minimum amount of redundancy possible in a fixed set of variable-length codes and does not mean that Huffman encoding is an optimal encoding method. It means that it provides the best approximation for encoding symbols when using fixed-width codes.

The entropy of a data file is given by Equation 1. In general, $log_2(p_i)$ will not be an integer so that the achieved data rate exceeds H bits/pixel; however, the data rate can be made to asymptotically approach H with increasing block size. Huffman encoding operates similar to the Morse code in which the most frequently used symbols are given the shortest code. However, it is not obvious how to optimally assign a set of varying length codes to a set of symbols to be transmitted. Huffman encoding optimizes by assigning a varying length code to a set of symbols based on the probability of occurrence. This method does not exclude the possibility of other lossless techniques producing a higher compression rate.

Suppose our total set of characters is represented by eight ASCII values with probabilities in Table 1. Eight ASCII values can be represented by 3 bits/pixel; however, for the given probabilities of occurrence, the entropy is only 2.6984 bits/pixel. Therefore, Huffman encoding can provide a compression scheme with a bit rate approaching 2.6984 bits/pixel.

Table 1
Sample 3-Bit Character Set with Probabilities of Occurrence

Character #	Probability	Entropy (H)
1	0.30	0.521089 = -0.3 $\log_2(0.3)$
2	0.23	0.487667 = -0.23 $\log_2(0.23)$
3	0.15	0.410544 = -0.15 $\log_2(0.15)$
4	0.08	0.291508 = -0.08 $\log_2(0.08)$
5	0.06	0.243533 = -0.06 $\log_2(0.06)$
6	0.06	0.243533 = -0.06 $\log_2(0.06)$
7	0.06	0.243533 = -0.06 $\log_2(0.06)$
8	0.06	0.243533 = -0.06 $\log_2(0.06)$
	1.0	Total entropy = 2.6984 bits/pixel

The Huffman code is a variable length code in which no code can be a prefix for any other Huffman code. Thus, unambiguous decoding of the transmitted data stream is allowed.

The Huffman codebook that is generated by the tree algorithm must be transmitted prior to the actual data in order to set up the decoder. Programs for Huffman encoding require two passes over the data. Probabilities are

generated in the first pass, and the actual Huffman compressed file is generated during the second pass. The following is an outline of the Huffman coding algorithm:

a. Arrange the symbol probabilities p_i in a decreasing order, and consider them as leaf nodes of a tree.

b. While there is more than one node:

(1) Merge the two nodes with smallest probability to form a new node whose probability is the sum of the two merged nodes.

(2) Arbitrarily assign 1 and 0 to each pair of branches merging into a node.

c. Read sequentially from the root node to the leaf node where the symbol is located.

The actual achieved compressed data rate is 2.71 bits/pixel and can be determined as follows:

0.30 * 2 bits = 0.60
0.23 * 2 bits = 0.46
0.15 * 3 bits = 0.45
0.08 * 3 bits = 0.24
0.06 * 4 bits = 0.24
0.06 * 4 bits = 0.24
0.06 * 4 bits = 0.24
0.06 * 4 bits = 0.24
Total = 2.71 bits/symbol

The coding sequence from Figures 1 and 2 follows:

Character #	Probability	Huffman Codeword
1	0.30	10
2	0.23	00
3	0.15	110
4	0.08	010
5	0.06	0110
6	0.06	0111
7	0.06	1110
8	0.06	1111

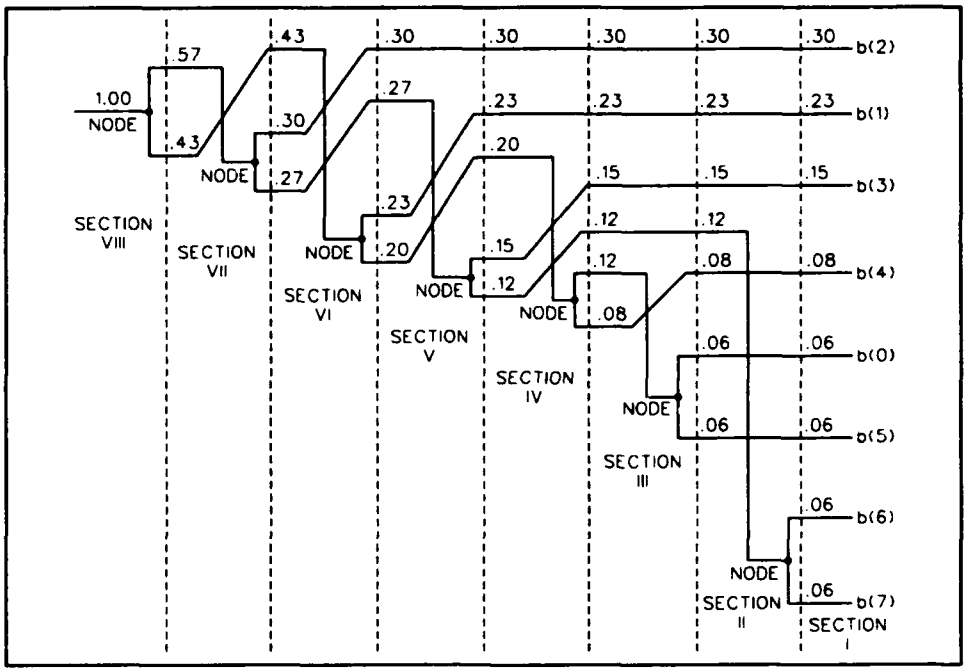

Figure 1. Construction of a binary Huffman code proceeds from right to left. At each section the tow bottom-most branches are combined to form a node and followed by a reordering of probabilities into descending order. These probabilities are then used to start the next section

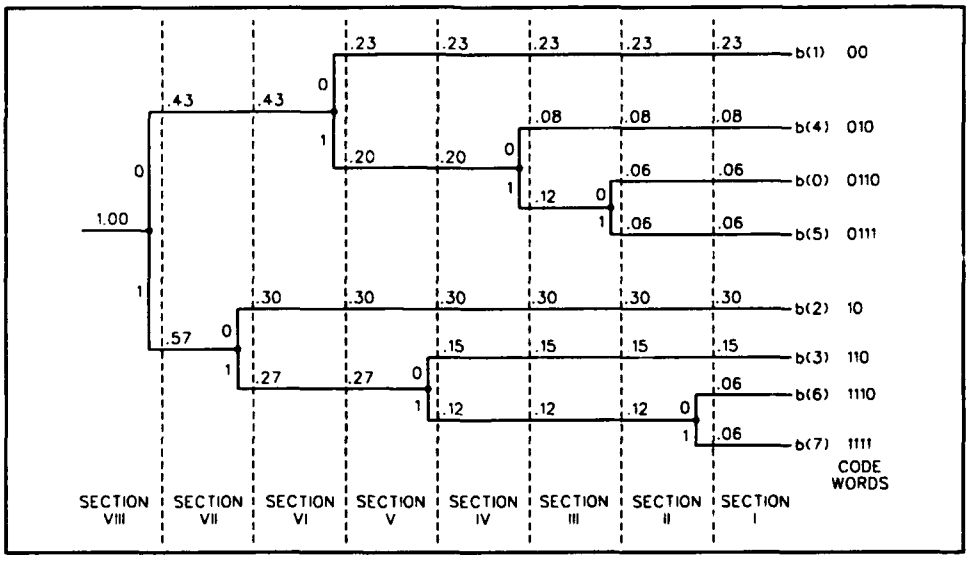

Figure 2. After the code construction, the tree is rearranged to eliminate crossovers, and coding proceeds from left to right. At each node a step-up produces a zero and a step-down a one. Resulting code words are shown at the right for each of the eight levels. Note that no code word is a prefix of any other code word

In high performance data compression, Huffman encoding faces a significant problem. The program has to pass a complete copy of the coding statistics to the expansion program. For a Huffman Order-0 encoder, there are 8 bits/byte, and the probability table that is passed to the decoder may occupy as little as 256 bytes. However, a Huffman Order-0 encoder sometimes cannot achieve compression rates close to the entropy given in Equation 1. A Huffman Order-1 encoder groups the symbols into 16-bit words but boosts the statistics table from 256 to 65,536 bytes. Though compression ratios will undoubtedly improve when moving to order-1, the overhead of passing the statistics table will probably nullify any gains.

Adaptive Huffman encoding solves this situation without paying any penalty for added statistics. It does so by adjusting the Huffman tree based on data previously seen, with no knowledge about future statistics. When using an adaptive model, the pixel information does not have to be scanned once in order to generate statistics. Instead, the statistics are continually modified as new characters are read in and encoded. A more detailed discussion of Adaptive Huffman coding can be obtained from a study by Nelson (1991).

Arithmetic Coding

Arithmetic coding is a lossless technique that can compress below the entropy level. It generally provides 10 to 20 percent better compression than Huffman. The output from an arithmetic process is a single number less than 1 and greater than or equal to 0. This single number can be uniquely decoded to exactly recreate the original byte stream.

To construct the output number, the symbols are assigned a set of probabilities. The message "KING HENRY" would have a probability distribution as follows:

Character	Probability
Space	1/10
E	1/10
G	1/10
H	1/10
I	1/10
K	1/10
N	2/10
R	1/10
Y	1/10

Individual characters can now be assigned along a probability line as follows:

Character	Probability	Range
Space	1/10	$0.00 \geq r > 0.10$
E	1/10	$0.10 \geq r > 0.20$
G	1/10	$0.20 \geq r > 0.30$
H	1/10	$0.30 \geq r > 0.40$
I	1/10	$0.40 \geq r > 0.50$
K	1/10	$0.50 \geq r > 0.60$
N	2/10	$0.60 \geq r > 0.80$
R	1/10	$0.80 \geq r > 0.90$
Y	1/10	$0.90 \geq r > 1.00$

The most significant portion of an arithmetic encoded message belongs to the first symbol, or "K," in the message "King Henry." To decode the first character properly, the final encoded message has to be a number greater than or equal to 0.5, and less than 0.6. Therefore, the low end for this range is 0.5, and the high end is 0.6. During the rest of the encoding process, each new symbol will further restrict the possible range of the output number. The next character to be encoded, the letter "I," has a range from 0.4 to 0.5 in the subrange 0.5 to 0.6. Applying this logic will further restrict the number to the range of 0.54 to 0.55. Continuing this process results in the final low value of 0.5464063556, which will uniquely decode into "KING HENRY," as tabulated below.

New Character	Low Value	High Value
K	0.5	0.6
I	0.54	0.55
N	0.546	0.548
G	0.5464	0.5466
Space	0.54640	0.54642
H	0.546406	0.546408
E	0.5464062	0.5464064
N	0.54640632	0.54640636
R	0.546406352	0.546406356
Y	0.5464063556	0.546406356

Arithmetic coding is best accomplished using 32-bit binary integral math. The decimal point is implied at the left side of the word. The initial value of HIGH is $FFFFFFFF, and LOW is $00000000. Decoding is accomplished by first transmitting a table of statistics to the decoder and then by transmitting a series of integers to be decoded. Typically, arithmetic coding is not used in shareware since IBM holds the patent for this technique.

Lempel, Ziv, Welch (LZW) Compression

The LZW compression is related to two compression techniques known as LZ77 and LZ78. LZ77 is a "sliding window" process in which the dictionary consists of a set of fixed-length phrases found in a "window" in the previously processed text. The size of the window is generally somewhere between 2-K and 16-K bytes, with the maximum phrase length ranging from perhaps 16 to 64 bytes. LZ78 takes a completely different approach to building a dictionary. Instead of using fixed-length phrases from a window in the text, LZ78 builds phrases up one symbol at a time, adding a new symbol to an existing phrase when a match occurs.

The LZ77 technique has a major performance bottleneck. When encoding, it has to perform string comparisons against the look-ahead buffer for every position in the text window. As LZ77 tries to improve compression performance by increasing the size of the window (and thus the dictionary), this performance bottleneck only gets worse. LZSS seeks to avoid some of the performance problems in the LZ77 algorithm. Under LZ77, the phrases in the text window were stored as a single contiguous block of text, with no other organization on top of it. LZSS still stores text in contiguous windows, but it creates an additional data structure that improves on the organization of the phrases. As each phrase passes out of the look-ahead buffer and into the encoded portion of the test windows, LZSS adds the phrase to a tree structure. The savings created by using the tree not only make the compression side of the algorithm much more efficient but also encourage experimentation with longer window sizes. Doubling the size of the text window now might only cause a small increase in the compression time; whereas, before it would have doubled it.

The LZ78 technique is similar to LZ77 but abandons the concept of a text window. Under LZ77, the dictionary of phrases was defined by a fixed window of previously seen text. With LZ78, the dictionary is a potentially unlimited list of previously seen phrases. An improved LZ78 algorithm in which the phrase dictionary is preloaded with single-symbol phrases equal to the number of symbols in the alphabet is LZW. However, LZW never outputs single characters, only phrases.

The LZW compression is shown using "BET BE BEE BED BEG" as the input stream. The quotation marks are used only to indicate that the character stream begins with a "space." The first 256 codes (0 to 255) are

not shown since the following tabulation is generated by LZW as an appendum to the first 256 characters.

Step	Character	Code Output	New Code
1	Space	none	[already in table]
2	B	ASCII for space	space B at (256)
3	E	ASCII for B	BE at (257)
4	T	ASCII for E	ET at (258)
5	Space	ASCII for T	T space at (259)
6	B	none	[space B already in table]
7	E	256	space BE at (260)
8	Space	ASCII for E	E space at (261)
9	B	none	[space B already in table]
10	E	none	[space BE already in table]
11	E	260	BEE space (262)
12	Space	none	[E space already in table]
13	B	261	E space B at (263)
14	E	none	[BE already in table]
15	D	257	BED at (264)
16	Space	D	D space at (265)
17	B	none	[space B already in table]
18	E	none	[space BE already in table]
19	T	260	BET at (266)
20	<EOF>	ASCII for T	

The first seven steps in this sequence are explained as follows:

Step 1: A "space" is read. The ASCII value for "space" is already contained in the table. No action is taken.

Step 2: A "B" is read. There is no "space B" combination in the table. The ASCII value for "space" is output, and "space B" is assigned code 256.

Step 3: An "E" is read. There is no "BE" combination in the table, so "BE" is assigned code 257 and the ASCII value for "B" is output.

Step 4: A "T" is read. There is no "ET" combination in the table, so "ET" is assigned code 258 and the ASCII value for "E" is output.

Step 5: A "space" is read. There is no "T space" combination in the table, so "T space" is assigned code 259 and the ASCII value for "T" is output.

Step 6: A "B" is read. The combination "space B" already exists in the table, so no action is required.

Step 7: An "E" is read. The combination "space BE" is assigned code 260, and code 256 for "space B."

The LZW decompressor takes the stream of codes from the compression algorithm and uses them to recreate the exact input stream. One reason for the efficiency of the LZW algorithm is that it does not need to pass the dictionary to the decompressor since the table can be built exactly as it was during compression, using the input stream as data. This step is possible because the compression algorithm always outputs the phrase and character components of a code before it uses it in the output stream, so the compressed information is not burdened with carrying a large dictionary. Typically 12-bit codes words will be used in an LZW algorithm for 4,096 possible phrases to accommodate the standard 256 eight-bit character set plus the additional phrases that are constructed as the input stream is processed. The patent for LZW is assigned to Unisys, which has made public its intention to protect its intellectual property rights. The LZW compression is defined as part of the CCITT V.42 bis specification, and Unisys has defined specific terms under which it will license the algorithm to modem manufacturers. It has not stated that it will apply the same terms to any parties manufacturing other types of products.

Lossless Compression Tests Results

At the U.S. Army Engineer Waterways Experiment Station (WES), tests were run on six lossless compression routines. The results of the tests are given in Table 2. The compression time and performance is given for each file and package. The first three files are executable programs. The next two are text files. JBARB2Y.COL and JGOLDY.COL are 8-bit grayscale images. The last four files are 8-bit RGB color images. The source and compiled codes for the following programs are available from a floppy disk entitled "Software listings" (1991). ARC, ZIP, and COMPRESS implement variations of the LZW algorithm. The highest performance package (in terms of compression) is LHARC by

Table 2
Lossless Compression Test Results

FILE	ARC	LHARC	ZOO	ZIP	COMPRESS	ENTROPY
WP.EXE	2.55s	6.35s	5.11s	3.53s	23.55s	
244,736	222,275	175,828	223,621	176,564	230,025	222,383
100%	90.8%	71.8%	91.4%	72.1%	94.0%	90.8%
WS.EXE	1.78s	2.40s	2.20s	1.34s	5.84s	
78,336	69,158	60,704	75,191	61,982	74,007	69,214
100%	88.3%	77.5%	96.0%	79.1%	94.5%	88.4%
PROCOMM.EXE	1.85s	4.60s	2.78s	2.90s	11.30s	
165,296	104,312	82,072	103,510	80,782	105,895	129,729
100%	63.1%	49.7%	62.6%	48.9%	64.1%	78.5%
SILVERD.DOC	7.18s	35.17s	11.29s	24.71s	62.44s	
1,096,064	403,771	365,080	412,111	359,696	363,231	586,292
100%	36.8%	33.3%	37.6%	32.8%	33.1%	53.5%
REGISTER.TXT	0.66s	0.66s	0.69s	0.55s	1.00s	
6,801	3,478	2,903	3,645	2,946	3,476	4,004
100%	51.1%	42.7%	53.6%	43.2%	51.1%	58.9%
BARB2Y.COL	6.40s	11.71s	10.59s	5.92s	38.78s	
403,200	377,351	352,790	412,625	373,014	381,029	377,173
100%	93.6%	87.5%	102%	92.5%	94.5%	93.5%
GOLDY.COL	4.40s	11.03s	7.53s	6.40s	36.9s	
403,200	373,323	339,233	372,283	350,122	350,123	380,240
100%	92.5%	84.1%	92.3%	86.8%	86.8%	94.3%
BARB.RGB	3.21s	13.51s	4.28s	9.54s	25.6s	
403,200	150,193	152,410	152,041	165,471	146,998	248,399
100%	37.3%	37.8%	37.7%	41.0%	36.5%	61.6%
GOLD.RGB	2.73s	14.78s	3.84s	13.73s	20.72s	
403,200	122,381	124,775	119,857	135,986	119,843	235,154
100%	30.3%	30.9%	29.7%	33.7%	29.7%	58.3%
BOATS.RGB	2.65s	19.79s	3.53s	14.87s	17.92s	
403,200	94,844	98,351	95,231	106,958	96,405	204,358
100%	23.5%	24.4%	23.6%	26.5%	23.9%	50.6%
BOARD.RGB	2.67s	15.93s	3.59s	14.29s	19.09s	
403,200	107,201	108,918	107,458	118,593	105,775	218,287
100%	26.6%	27.0%	26.7%	29.4%	26.2%	54.1%

Haruyasu Yoshizaki which is built on an LZ77 algorithm using a dictionary-based method on a sliding window that moves through the text. All of these packages are compared against standard entropy coding (Table 2) which consistently provided the worst compression.

Under each routine in Table 2, compression time is shown in seconds for an IBM compatible 486/33Mhz PC. The next two rows show the file size after compression and the size of the compressed file as a percentage of the original.

Exact Coding of Difference Files

If a pixel by pixel difference file is created from an original image, then Huffman encoding can be used to give a lossless compression at ratios that can exceed the other lossless techniques in Table 2. The first pixel was retained as its original value, and all the other 8-bit values represent the differences between the previous pixel and the present pixel. The following code section shows how overflow programming can avoid 9-bit differences:

```
 90 a$ = INPUT$(1,1)
110 a = ASC(a$) - 128
115 c = a - olda
123 IF c > 127 THEN c = c - 256
124 IF c < -128 THEN c = 256 + c
130 PRINT #2, CHR$(c + 128);
140 olda = a
150 GOTO 90
```

File (403,200)	Compressed Size
JBARB2Y.COL	300,681
JGOLDY.COL	257,544
BARB.RGB	164,107
GOLD.RGB	125,458
BOATS.RGB	106,120
BOARD.RGB	127,346

2 Lossy Compression Techniques

The Fourier Transform

Lossy compression methods are useful for image compression and are often based on the Fourier transform. In general, a two-dimensional (2-D) Fourier transform of an image will produce a large number of coefficients close to zero. For example, if 95 percent of the transform coefficients can be approximated by zero, then the image may be adequately represented using only 5 percent of the Fourier coefficients. The sparse matrix that results is usually Huffman encoded. Although LZW and Arithmetic codes generally provide more compression than Huffman, the latter tends to work better on sparse data files that contain a large number of common values.

A 2-D Fourier Transform is required for image files since images are 2-D. The 2-D transform is equivalent to taking the 1-D transform of each row, and then taking the 1-D transform of each column, for an image matrix. The problem with the Fourier Transform is that it leaves us with a real component and an imaginary component for each pixel in the image. By symmetrically extending the image before taking the 2-D Fourier transform, the imaginary, or sine, terms can be forced to zero.

Assume an image is represented by the 4x4 matrix, U, such that

$$[U] = \begin{bmatrix} 127 & 123 & 119 & 110 \\ 115 & 11 & 103 & 98 \\ 99 & 87 & 85 & 83 \\ 94 & 82 & 81 & 79 \end{bmatrix}$$

The 2-D Fourier Transform will produce 32 terms, or a real term (cosine) and an imaginary term (sine) for each of the 16-pixel values in the matrix. By symmetrically extending U, either the cosine terms or the sine terms can be forced to be identically zero.

A symmetrical extension of U given by

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 127 & 123 & 119 & 110 & 119 & 123 & 127 \\ 0 & 115 & 111 & 103 & 98 & 103 & 111 & 115 \\ 0 & 99 & 87 & 85 & 83 & 85 & 87 & 99 \\ 0 & 94 & 82 & 81 & 79 & 81 & 82 & 94 \\ 0 & 99 & 87 & 85 & 83 & 85 & 87 & 99 \\ 0 & 115 & 111 & 103 & 98 & 103 & 111 & 115 \\ 0 & 127 & 123 & 119 & 110 & 119 & 123 & 127 \end{bmatrix}$$

will produce only nonzero cosine coefficients under a 2-D Fourier Transform.

This modification of the Fourier Transform is called a Cosine Transform and allows the matrix U to be fully represented by 16 unique coefficients, most of them close to zero. If a 2-D Fast Fourier Transform is performed on the symmetrically extended matrix, the results are equivalent to a Fast Cosine Transform. It is important to note that the coefficients produced by a Cosine Transform are not the same as the real part of the Fourier Transform applied to the original matrix, U.

An antisymmetric extension of U given by

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 127 & 123 & 119 & 110 & -127 & -123 & -119 & -110 \\ 0 & 115 & 111 & 103 & 98 & -115 & -111 & -103 & -98 \\ 0 & 99 & 87 & 85 & 83 & -99 & -87 & -85 & -83 \\ 0 & 94 & 82 & 81 & 79 & -94 & -82 & -81 & -79 \\ 0 & -127 & -123 & -119 & -110 & 127 & 123 & 119 & 110 \\ 0 & -115 & -111 & -103 & -98 & 115 & 111 & 103 & 98 \\ 0 & -99 & -87 & -85 & -83 & 99 & 87 & 85 & 83 \\ 0 & -94 & -82 & -81 & -79 & 94 & 82 & 81 & 79 \end{bmatrix}$$

will produce only nonzero sine coefficients under a 2-D Fourier Transform. This modification of the Fourier Transform is called a Sine Transform.

Representation of 8-Bit Images

The images used in this article are raw 8-bit grayscale images. The word "raw" is used to enforce the idea that there is no header on the file. The test image used for lossy compression, DBSJ.4, contains 448 columns and 280 rows for a total of 125,540 bytes. Each byte controls the grayscale intensity of a pixel on the screen with 0 representing black and 255 representing white, for a total of 256 shades of gray. A separate palette file controls whether the image is grayscale or color. The palette files used with the program "Image" consist of 768 bytes, or 256 bytes for the color red, 256 bytes for green, and 256 bytes for blue. If a grayscale image is desired, then the palette "gray.pal" is used and includes identical 256-byte segments for red, green, and blue. The actual computation of the color of each pixel in an RGB image can be seen easily in line 380. For

color images, bits 5, 6, and 7 determine the intensity of the red component; bits 2, 3, and 4 define the green intensity; and bits 0 and 1 provide four levels of blue. The palette file defines the mapping of the red, green, and blue bits to various intensity levels. VGA (Video graphic adapter) mode 13 is set up in line 220 for 320x200 resolution with 256 colors, or shades of gray, depending on the palette that is selected. The program "Image" is useful for displaying an image prior to compression, and then showing the degradation caused by a lossy compression/decompression routine.

While the program "Image" can let the user visually determine the quality of a reconstructed image by comparison to the original, the program "MSE" computes the mean-square-error between the original image and the reconstructed image to provide an analytical measure of quality. The mean-square-error is defined as

$$MSE = \frac{1}{N} \sum_{i=1}^{N} \left(X_i - X_i' \right)^2 \tag{2}$$

where X_i is the i^{th} pixel in the original image, X_i' is the i^{th} pixel in the reconstructed image, and N is the number of pixels. In general, a *MSE* of five or less indicates very little loss of detail.

The Discrete Cosine Transform (DCT)

The DCT is simply a separate mathematical method for generating the Cosine Transform without explicitly using the Fourier Transform. It is based on defining a matrix [*C*] such that

$$C(k,n) = \frac{1}{\sqrt{N}} \text{ for } k,n=0$$

$$C(k,n) = \sqrt{\frac{2}{N}} \cos\left[\pi \cdot (2 \cdot k + 1) \cdot \frac{n}{2 \cdot N} \right] \textit{ otherwise}$$

If an image is divided into blocks of size 8x8, then the matrix *C* must also be 8x8 and becomes

$$[C] = \begin{bmatrix} 0.35355 & 0.35355 & 0.35355 & 0.35355 & 0.35355 & 0.35355 & 0.35355 & 0.35355 \\ 0.49039 & 0.41573 & 0.27778 & 0.0975 & -0.0975 & -0.27778 & -0.41573 & -0.49039 \\ 0.46193 & 0.19134 & -0.19134 & -0.46193 & -0.46194 & -0.19134 & 0.19133 & 0.46193 \\ 0.41573 & -0.09754 & -0.49039 & -0.27778 & 0.27778 & 0.49039 & 0.09754 & -0.41573 \\ 0.35355 & -0.35355 & -0.35355 & 0.35355 & 0.35355 & -0.35355 & -0.35355 & 0.35355 \\ 0.27778 & -0.49039 & 0.09754 & 0.41573 & -0.41573 & -0.09754 & 0.49039 & -0.27778 \\ 0.19134 & -0.46194 & 0.46193 & -0.19133 & -0.19134 & 0.46194 & -0.46193 & 0.19133 \\ 0.09754 & -0.27778 & 0.41573 & -0.49039 & 0.49039 & -0.41573 & 0.27777 & -0.09754 \end{bmatrix}$$

If a matrix U represents the pixels in an 8x8 block of an image, then the DCT is define by the equation

$$[V] = [C]\,[U]\,[C^T] \tag{3}$$

using matrix multiplication and $[C^T]$ as the transpose of matrix $[C]$. $[V]$ becomes a sparse matrix with few large coefficients. For example,

$$let\ [U] = \begin{bmatrix} 48 & 48 & 51 & 53 & 54 & 54 & 57 & 62 \\ 50 & 51 & 52 & 52 & 57 & 59 & 59 & 63 \\ 51 & 52 & 55 & 56 & 59 & 61 & 62 & 67 \\ 53 & 55 & 57 & 57 & 62 & 64 & 63 & 66 \\ 56 & 58 & 57 & 60 & 65 & 67 & 68 & 68 \\ 56 & 59 & 61 & 63 & 66 & 66 & 70 & 72 \\ 60 & 61 & 64 & 65 & 67 & 70 & 72 & 72 \\ 62 & 62 & 65 & 70 & 69 & 70 & 73 & 76 \end{bmatrix}$$

which represents the first 8x8 subblock in the grayscale test file DBSJ.4. The actual transform file, V, is rounded to integer values and becomes

$$[V] = \begin{bmatrix} 61 & -5 & 0 & 0 & 0 & 0 & 0 & 0 \\ -5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

which contains a large number of zeroes, but retains almost all of the information in the original subimage $[U]$. Entropy analysis of the original matrix $[U]$ reveals that a compression of 1.77 to 1 is possible using Huffman encoding (as shown by running ENTROPY.EXE). However, a compression of 25.6 to 1 is possible using Huffman encoding of the matrix $[V]$. This method is the basis of the Joint Photographic Expert Group (JPEG) algorithm in which images are divided into 8x8 subblocks; each subblock is processed independently by a DCT algorithm, and then the resulting file is Huffman encoded.

The inverse DCT can be used to regenerate an approximation to the matrix $[U]$ using the equation

$$[U'] = [C^T][V][C] \tag{4}$$

where $[U']$ represents the reconstructed image. Using the inverse DCT, the matrix $[U']$ becomes

$$[U'] = \begin{bmatrix} 47 & 48 & 50 & 53 & 55 & 58 & 60 & 61 \\ 48 & 49 & 51 & 54 & 57 & 59 & 61 & 62 \\ 50 & 51 & 53 & 56 & 58 & 61 & 63 & 64 \\ 53 & 54 & 56 & 58 & 61 & 64 & 65 & 67 \\ 55 & 57 & 58 & 61 & 64 & 66 & 68 & 69 \\ 58 & 59 & 61 & 64 & 66 & 69 & 71 & 72 \\ 60 & 61 & 63 & 65 & 68 & 71 & 73 & 74 \\ 61 & 62 & 64 & 67 & 69 & 72 & 74 & 75 \end{bmatrix}$$

which is nearly identical to the original image $[U]$ and differs by a mean-square-error of 1.85 indicating that visual differences will not be perceptible. A compression scheme using the DCT can be summarized in the following steps:

a. Divide the image into 8x8 blocks.

b. Perform the DCT on each 8x8 block.

c. Replace values close to zero with zero.

d. Huffman encode the resulting file.

Decompression involves the following:

a. Huffman decode the compressed file.

b. Perform the inverse DCT on each 8x8 block.

The entire DBSJ.4 image is shown in Figure 3 before and after compression at a rate of 25:1. The programs DCT.BAS and INVDCT.BAS were used for the transformations. The final compression of 25:1 was achieved by truncating any DCT coefficients less than five to zero and then entropy encoding of the remaining DCT coefficients.

The JPEG algorithm is basically an 8x8 DCT with some enhancements. JPEG is generally considered feasible for compression of still images in the 10:1 to 25:1 range. Another standard, Motion Picture Expert Group (MPEG), is used for motion video and exploits the images differences frame-by-frame. MPEG is considered feasible for the compression of real-time video images at the rate of 50:1.

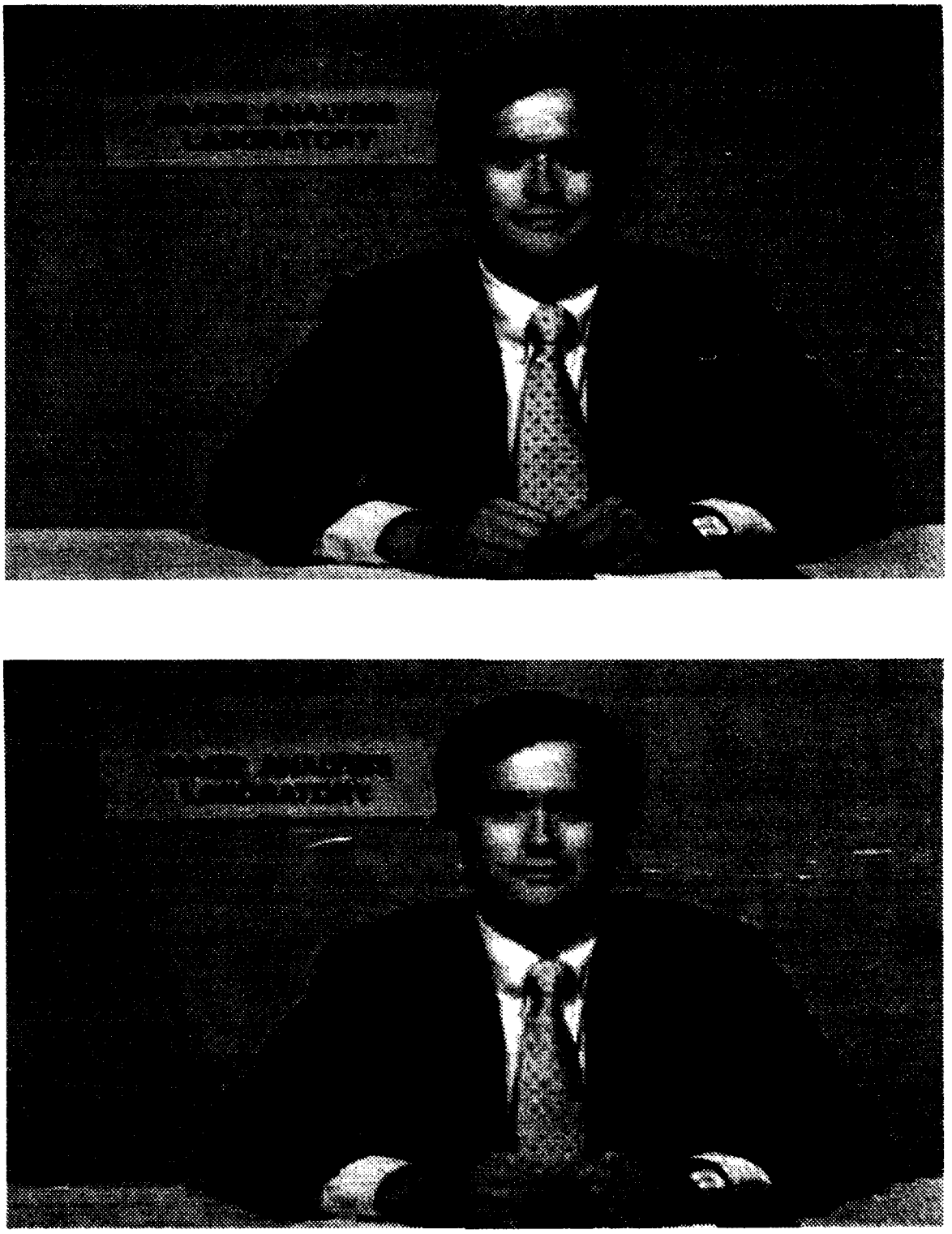

Figure 3. These photographs show the original image (top), and the restored (bottom) after 25:1 compression (MSE = 25.39 using a threshold level of 5 in the DCT coefficient matrix)

The program, DCT.BAS, performs the DCT on any grayscale image file. *X* and *Y* dimensions are required as input parameters. The output file is given the extension .DCT.

The .DCT file is a binary file. The user can perform Huffman encoding directly on this file to achieve a high level of compression. Even higher levels of compression can be obtained by defining a threshold level in the .DCT file such that coefficients below the threshold are replaced by zero before Huffman encoding.

An inverse DCT can be obtained using INVDCT.BAS. This program is very similar to DCT.BAS. The major change is that the indices of matrix [*C*] are swapped in lines 630 and 730. Since matrix [*C*] is orthogonal, the transpose of [*C*] is identical to the inverse of [*C*]. The quality of the restoration depends on the threshold level set previously for the truncation of the DCT coefficients. These two programs will also handle other types of transforms including Hadamard and Sine transforms simply by modifying the subroutine that defines the transform matrix [*C*]. The Hadamard and Sine transforms do not provide as much compression as the Cosine transform, so their discussion is presented in a study by Jain (1989).

JPEG and MPEG

While the JPEG algorithm was defined for still images and is based on the DCT, MPEG is meant primarily for motion video and obtains additional compression by exploiting interframe redundancy. Basically, if an *NxN* pixel block changes very little in successive frames, then the DCT coefficients saved from a previous frame can be reused to represent the N^2 pixel values. The basis for many proposed HDTV systems is MPEG.

The JPEG algorithm processes 8-bit color RGB (red, green, blue) images by first translating from RGB to Y-C_b-C_r. For example, an original 512x512 eight-bit RGB color image contains 262,144 bytes. The three most significant bits define the *RED* intensity. The next three bits define the *GREEN* intensity, and the lowest two bits determine the *BLUE* intensity. In preparation for Y-C_b-C_r conversion, three temporary 512x512 files are created including a 512x512 file for the *RED* component which consists of the three most significant bits (MSBs) corresponding to red padded with five zeroes, a 512x512 file for the *GREEN* component which consists of the three bits for green moved to the most significant position and padded with five zeroes, and a 512x512 file for the *BLUE* component which consists of the two bits for blue moved to the most significant position and padded with six zeroes. The translation format from RGB to Y-C_b-C_r is shown as

$$Y = 0.299 \cdot RED + 0.587 \cdot GREEN + 0.114 \cdot BLUE$$

$$C_b = -0.16874 \cdot RED - 0.33126 \cdot GREEN + 0.5 \cdot BLUE$$

$$C_r = 0.5 \cdot RED - 0.41869 \cdot GREEN - 0.08131 \cdot BLUE$$

Each of the resulting Y, C_b, and C_r files are also 512x512 eight-bit files. The Y component (luminance) contains the majority of the information while the bandwidths of the C_b and C_r components are relatively small by comparison and can therefore be compressed at a much higher ratio without loss of information. A DCT can now be done separately on the Y, C_b, and C_r files. RGB format is regained with the equations

$$RED = Y + 1.402 \cdot C_r$$

$$GREEN = Y - 0.34414 \cdot C_b - 0.71414 \cdot C_r \qquad (5)$$

$$BLUE = Y + 1.772 \cdot C_b$$

The JPEG algorithm is actually a modified DCT that includes DCPM encoding of the DC coefficients, a zig-zag method of writing out the DCT coefficients, the definition of a "quality factor" that defines the quantization level of the DCT coefficients, quantization according to the JPEG visualization matrix, and Huffman encoding of the resulting file. The reader is referred to Nelson (1991) for complete details on JPEG.

Singular Value Decomposition (SVD)

Singular Value Decomposition (SVD) is useful mainly as a gauge against which to measure the performance of other transform techniques and in image restoration. The SVD concentrates the maximum amount of energy in the fewest eigenvalues and is optimal in the least square sense.

Let $[U]$ be an image matrix. The matrices $[U][U^T]$and $[U^T][U]$ are nonnegative and symmetric and have the identical eigenvalues, $\{\lambda_m\}$. Assuming that U is an NxN matrix of N^2 pixels, there are at most $r \geq N$ nonzero eigenvalues. It is possible to find r orthogonal $Nx1$ eigenvectors $\{\Phi_m\}$ of $[U^T][U]$, and r orthogonal $Nx1$ eigenvectors $\{\Psi_m\}$ of $[U][U^T]$, that is

$$[U^T][U]\Phi_m = \lambda_m \Phi_m, \; m=1,.....,r \qquad (6)$$

$$[U][U^T]\Psi_m = \lambda_m \Psi_m, \; m=1,.....,r \qquad (7)$$

The matrix U has the representation

$$[U] = \Psi \Lambda^{0.5} \Phi^T = \sum_{m=1}^{r} \Psi_m \Phi_m^T \sqrt{\lambda_m} \tag{8}$$

where Ψ and Φ are Nxr matrices whose m^{th} columns are the vectors Ψ_m and Φ_m, respectively, and $\Lambda^{0.5}$ is an rxr diagonal matrix, defined as

$$\Lambda^{0.5} = \begin{bmatrix} \sqrt{\Gamma_1} & & \\ & \ddots & \\ & & \sqrt{\Gamma_m} \end{bmatrix} \tag{9}$$

Equation 8 is called the spectral representation, the outer product expansion, or the singular value decomposition (SVD) of $[U]$. The nonzero eigenvalues of $[U^T]$ $[U]$, Γ_m, are also called the singular values of $[U]$. If $r << N$, then the image containing N^2 samples can be represented by $(N + N)r$ samples of the vectors $\left\{\lambda_m^{1/4}\Psi_m, \lambda_m^{1/4}\Phi_m ; m=1, \ldots\ldots, r\right\}$.

Since Ψ and Φ have orthogonal columns, the SVD transform of the image U is defined as

$$[U] = \Psi \Lambda^{0.5} \Phi^T \tag{10}$$

which is a separable transform that diagonalizes the given image.

The image $[U_k]$ generated by the partial sum

$$[U_k] = \sum_{m=1}^{k} \sqrt{\lambda_m}\ \Psi_m \Phi_m^T,\ k \le r \tag{11}$$

is the best least squares rank-k approximation of $[U]$ if the λ_m are in decreasing order of magnitude. For any $k \le r$, the least squares error

$$\varepsilon_k^2 = \sum_{m=1}^{M} \sum_{n=1}^{N} [u(m,n) - u_k(m,n)]^2,\ k=1,2,\ldots\ldots,r \tag{12}$$

reduces to

$$\varepsilon_k^2 = \sum_{m=k+1}^{r} \Gamma_m \tag{13}$$

These equations show that the energy concentrated in the transform coefficients is maximized by the SVD transform for the given image. As an example,

$$Let\ [U] = \begin{bmatrix} 1 & 2 \\ 2 & 1 \\ 1 & 3 \end{bmatrix}$$

The eigenvalues of $[U^T][U]$ are found to be $\lambda_1 = 18.06$ and $\lambda_2 = 1.94$, which give $r = 2$, and the SVD transform of $[U]$ is

$$\Lambda^{0.5} = \begin{bmatrix} 4.25 & 0 \\ 0 & 1.39 \end{bmatrix}$$

The eigenvectors are found to be

$$\Phi_1 = \begin{bmatrix} 0.5019 \\ 0.8649 \end{bmatrix} \qquad \Phi_2 = \begin{bmatrix} 0.8649 \\ -0.5019 \end{bmatrix}$$

From above, Ψ_1 is obtained via

$$\Psi_m = \frac{1}{\sqrt{\lambda_m}} [U] \Phi_m$$

to yield

$$[U_1] = \sqrt{\lambda_1}\Psi_1\Phi_1^T = \begin{bmatrix} 1.120 & 1.94 \\ 0.953 & 1.62 \\ 1.549 & 2.70 \end{bmatrix}$$

as the best least squares rank-1 approximation of $[U]$. The energy concentrated in the K samples of SVD is greater than the energy concentrated in any K samples of the other transform methods.

Example: Let $[U]$ be an NxN images matrix. It is desired to use SVD to achieve a $N{:}r$ compression using only r of the largest eigenvalues. The $[U]$ matrix can then be approximated as

$$[U] = \begin{bmatrix} \Psi_{11} & \cdots & \Psi_{1r} & 0 & \cdots & 0 \\ \vdots & & \vdots & & & \\ \Psi_{n1} & \cdots & \Psi_{nr} & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} \sqrt{\lambda_1} & 0 & \cdots & & & 0 \\ 0 & \sqrt{\lambda_2} & & & & \\ \vdots & & \ddots & & & \\ & & & \sqrt{\lambda_r} & & \\ & & & 0 & & \\ 0 & & & & & 0 \end{bmatrix} \begin{bmatrix} \Phi_{11} & \Phi_{12} & \cdots & \Phi_{1n} \\ \vdots & & & \vdots \\ \Phi_{r1} & \Phi_{r2} & \cdots & \Phi_{m} \\ 0 & \cdots & & 0 \\ \vdots & & & \vdots \\ 0 & \cdots & & 0 \end{bmatrix}$$

The total number of coefficients that must be encoded and transmitted are rxN coefficients from matrix Ψ plus rxN coefficients from matrix Φ or $2xrxN$ coefficients total, which only achieves half of the desired compression. A small improvement can be obtained by recognizing that the columns

in Ψ and the rows in Φ are orthogonal. If Ψ has r orthogonal columns such that

$$\Psi = \begin{bmatrix} \Psi_{11} & \Psi_{12} & \cdot & \Psi_{1r} & 0 & \cdot & 0 \\ \cdot & & & & \cdot & & \cdot \\ \cdot & & & & \cdot & & \cdot \\ \cdot & & & & \cdot & & \cdot \\ \cdot & & & & \cdot & & \cdot \\ \Psi_{n1} & \Psi_{n2} & \cdot & \Psi_{nr} & 0 & \cdot & 0 \end{bmatrix}$$

with rxN nonzero elements, then the rN elements of Ψ can be reconstructed using the conditions for orthogonality from the matrix Ψ' with

$$\Psi' = \begin{bmatrix} 0 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 0 \\ \Psi_{11} & 0 & \cdot & \cdot & \cdot & \cdot & \cdot & 0 \\ \cdot & \Psi_{12} & 0 & \cdot & \cdot & \cdot & \cdot & 0 \\ \cdot & \cdot & & & & & & \cdot \\ \cdot & \cdot & & & & & & \cdot \\ \Psi_{n1} & \Psi_{n2} & \cdot & \cdot & \Psi_{nr-1} & 0 & \cdot & 0 \end{bmatrix}$$

where Ψ' has $rN - 1/2\ r^2$ nonzero elements. The number of elements that are required to be transmitted are $rxN - 1/2\ r_2$ for Ψ and $rxN - 1/2\ r2$ for Φ or $r(2N - r)$ coefficients total, as opposed to rxN. Therefore, a degradation factor, D, can be computed as

$$D = 2 - r'$$

where r' equals r'/N and r' represents the percent compression. Therefore, if an attempt is made to achieve 90 percent compression with SVD on a data file of size N^2 by using only $1/10^{th}$ of the eigenvalues, the actual size of the data file that has to be transmitted is $0.1 \times (2 - 0.1) \times N^2$, or 19 percent of the original size, so that a compression of only 81 percent is achieved.

Image compression using SVD is of limited use for three reasons:

a. Vast numbers of computations are required.

b. The degradation factor limits the actual compression to approximately one half of the theoretical limit.

c. Each coefficient in the Ψ and Φ matrices will require more that a single byte of accuracy.

Example: If the SVD transform of an image matrix, $[U]$, produces eigenvalues that are zero, then by not transmitting these zero eigenvalues and their corresponding zero rows and columns in Ψ and Φ^T, respectively, lossless image

compression can be achieved. If $[U]$ is an 256x256 matrix given by

$$[U] = \begin{bmatrix} 0,1,2,3,4,5,.....,255 \\ 0,1,2,3,4,5,.....,255 \\ 0,1,2,3,4,5,.....,255 \\ \vdots \\ 0,1,2,3,4,5,.....,255 \end{bmatrix}$$

then the SVD transform produces 255 eigenvalues that are zero and one eigenvalue that is nonzero. The (unnormalized) Ψ and Φ^T matrices become

$$\Psi^T = [1,1,1,1,.....1]$$

and

$$\Phi = [0,1,2,3,4,5,..,255]$$

The original U matrix can be reconstructed by simple matrix multiplication of Ψ and Φ^T.

All of the energy is compacted into a single eigenvalue for a theoretical possible compression of 1/256, but notice that a total of 512 bytes must be transmitted in order to send the 256-element Ψ matrix and the 256-element Φ matrix. Therefore, only a compression of 1/128 can be realized which is consistent with the degradation factor

$$D = \frac{1}{256} \cdot (2 - \frac{1}{256}) \approx 2$$

with all the energy compressed into 1/256 of the eigenvalues. Since the other 255 eigenvalues are exactly 0, the entire 65,536-element U matrix can be reconstructed exactly from 512 transmitted elements. Huffman encoding would not be able to achieve any compression on matrix $[U]$ since all of the possible ASCII values occur with equal probability.

Example: DBSJ.4 represents a 448 x 280 eight-bit grayscale image. The SVD of DBSJ.4 was taken using MATLAB. Figure 4 represents the image quality for a compression ratio of 100:1.

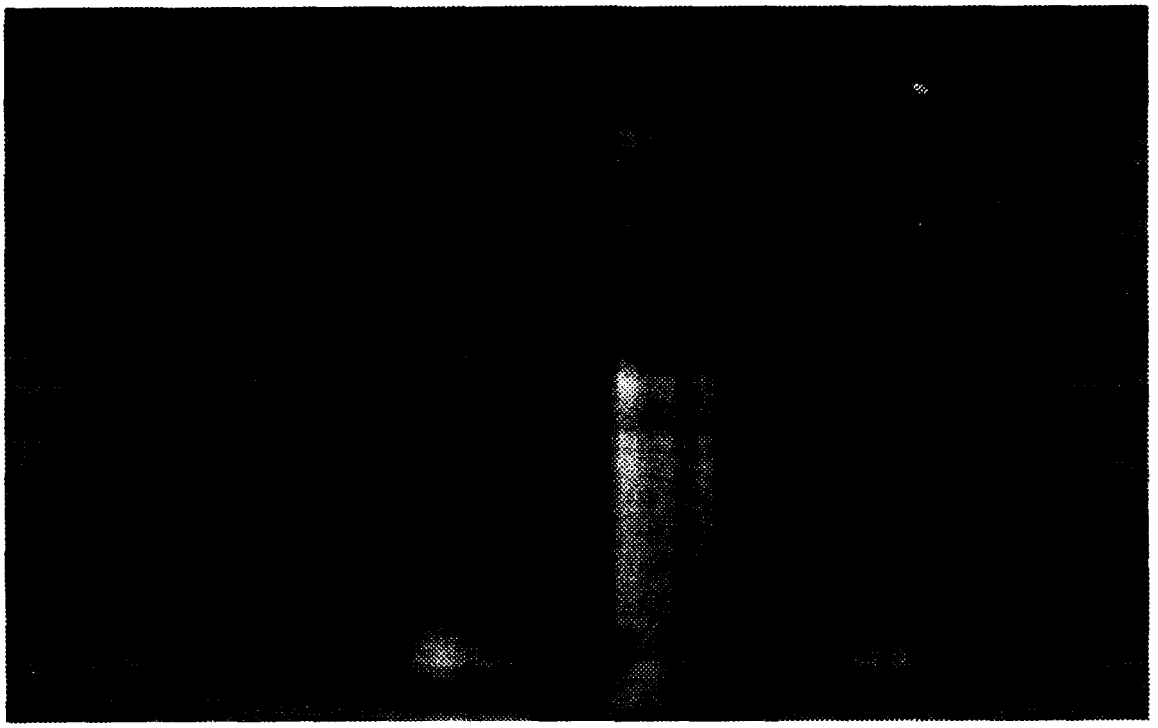

Figure 4. DBSJ.4 images compressed with SVD at 100:1 with a mean-square error of 553

Fractal Compression

Fractal compression is a new technique that promises compression ratios of 500:1 without noticeable loss in image quality. It is based on taking a portion of the image and reproducing the rest of the image with a set of affine transformations that consist of translations, rotations, and scaling. A set of iterated function system (IFS) codes defines the image. Once the IFS codes for an image have been determined, then a simple iterated procedure is used to regenerate the image.

The iterated function is defined by the IFS codes. Essentially, X and Y coordinates of the new pixel are computed from X and Y coordinates of the previous pixel. The IFS codes given below will illustrate the procedure. The program, IFS.BAS, will generate a solid rectangle from these IFS codes in a random and rather spectacular way.

IFS Codes for a Square						
A	B	C	D	E	F	P
0.5	0	0	0.5	1	1	0.25
0.5	0	0	0.5	50	1	0.25
0.5	0	0	0.5	1	50	0.25
0.5	0	0	0.5	50	50	0.25

From the IFS codes, new values of X and Y are computed from one of four sets of equations as follows:

a. Equation Set 1: $New\ X = AX + BY + 1$
$New\ Y = CX + DY + 1$

b. Equation Set 2: $New\ X = AX + BY + 50$
$New\ Y = CX + DY + 1$

c. Equation Set 3: $New\ X = AX + BY + 1$
$New\ Y = CX + DY + 50$

d. Equation Set 4: $New\ X = AX + BY + 50$
$New\ Y = CX + DY + 50$

The variable P indicates that the probability that any one set is used to compute the new X and Y coordinates is 0.25. After each iteration of one set of equations, the X and Y coordinates are plotted, and another set of equations is randomly chosen to compute the next point.

The procedure is illustrated graphically in Figure 5. The X and Y coordinates are scaled by 0.5 and either a 1 or 50 is added to the X and/or Y values. This procedure is repeated until the entire rectangle is painted.

Any image can be reduced to sets of IFS codes. Barnsley (1988) provides other IFS codes that can be used with IFS.BAS for more exotic images.

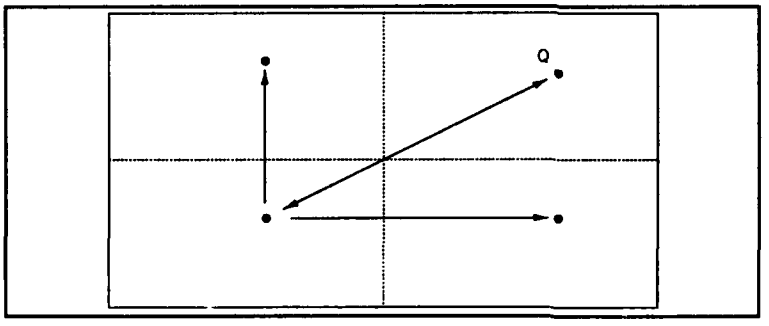

Figure 5. Random mapping of point Q for a fractal process

3 Image Compression and Bandwidth Reduction

Communication Standards

An analog NTSC cable TV system requires 6 MHz per channel. It might seem that a compression of 50:1 would reduce the bandwidth by a factor of 50 (to 120,000 Hz), but it does not. Using an estimated resolution of 512x480 for TV with 8-bit color and a frame rate of 30 times per second, the required bit rate for digital transmission is 58,982,000 bits/sec, or approximately 60 Mbps. Since compression techniques are inherently digital, the 6-MHz analog signal was converted to a 60,000,000-bit/sec digital signal and now requires an absolute minimum bandwidth of 30 MHz (based on 1 bit/symbol). Compression by 50:1 would result in a total transmission bandwidth of at least 600,000 KHz.

Shannon's law dictates the maximum transmission speed of digital data. This speed limit is determined by only two factors, which include the bandwidth of the transmission channel and the signal-to-noise ratio of the channel. In a digital system, the maximum data rate, called the channel capacity C, is bounded by Shannon's law which is given by

$$C\ bit/sec = BW \log_2 \left(1 + \frac{S}{N}\right)$$

where BW is the bandwidth of the channel in Hertz. According to this equation, if there is no noise in the channel ($S/N = infinity$), then the channel capacity becomes infinite regardless of the bandwidth.

As an example, consider an analog phone line with a bandwidth of 4,000 Hz, and assume that it is digitized at 8,000 times per second using 8 bits/byte. If no other noise is present on the line, the signal-to-noise ratio will be 48 dB since each bit of resolution in the A/D convertor decreases the quantization noise by 6 dB.

To use Shannon's law, the S/N ratio must be expressed in linear terms, or $S/N = 10^{(48/10)}$. The channel capacity then becomes

$$C = 64{,}000\ bits/sec = 4{,}000 \log_2 \left(1 + 10^{4.8}\right)$$

and will not be affected by the number of bits encoded in a symbol. The theoretical limit essentially states that it is possible to build dial-up modems that transmit without error at rates up to 64,000 bits/sec. Shannon's law does not tell us how to design hardware to achieve the theoretical maximum bit rate. The common quadrature phase shift keying (QPSK) and quadrature amplitude modulation (QAM) techniques used for modems have inherent deficiencies that will not allow them to obtain the maximum theoretical bit rate; however, Shannon's law can be used to determine the theoretical limit on the channel capacity.

The distinction between bit rate and symbol rate is vital in determining bandwidth requirements. The QPSK modulation technique encodes two bits in a symbol since there are four possible phases (0, 90, 180, and 270 deg) of the carrier. A change of phase of the carrier represents that transmission of a single symbol (baud) which conveys two bits of information according to Table 3:

Table 3
QPSK Modulation

Differential Phase	Dibits
0°	00
90°	01
180°	11
270°	10

Although it is common to speak of a 1,200 baud modem, the bit rate is 1,200 bits/sec and the baud rate (with QPSK modulation) is really 600 baud.

The QAM varies both the amplitude and phase of the carrier in order to encode bits into symbols. The term 16-QAM means the carrier can assume one of 16 possible states in order to encode $\log_2(16)$, or 4 bits/symbol. The constellation for a 16-QAM carrier is shown in Figure 6 where the distance from the origin represents the amplitude of the carrier, and the angle of the point represents the phase of the carrier.

The difficulty with using 64-QAM, 256-QAM, or higher is that the susceptibility to noise increases exponentially as the number of bits per symbol is increased.

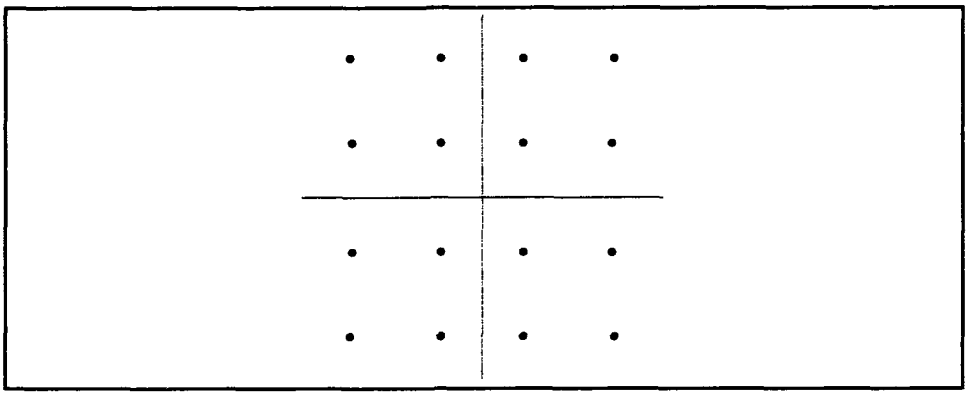

Figure 6. 16-QAM constellation

The bandwidth required for digital transmission of data is given by

$$BW = \frac{Data\ Rate\ (symbols/sec) \cdot (1 + r)}{2}$$

where r represents the Nyquist rolloff factor of the raised cosine filter and r falls between 0 and 1. The basis for this equation is given by Couch (1982).

In the previous example, if the 64,000-bit/sec data rate is to be transmitted using frequency shift keying (FSK) such that only one bit is encoded per symbol, then the symbol rate becomes 64,000 symbols/sec. The required transmission bandwidth is usually taken as 64,000 Hz (r = 1) in most texts, although it can approach as low as 32,000 Hz by using a sharp cutoff Nyquist filter (r = 0). Throughout the remainder of this report, it will be assumed that the required minimum transmission bandwidth is one half of the symbol rate.

Digital HDTV has a proposed resolution of 1,440x960 pixels/frame x 12 bits/pixel x 30 frames/sec (12 bits/pixel = 8 bits/pixel for luminance + 4 bits/pixel for chrominance at half-resolution). The data rate is about 500 Mbits/sec. The aim of most HDTV systems is to broadcast at about 15 Mbits/sec. This data rate requires approximately 33:1 compression. Encoding with 1 bit/symbol would require a transmission bandwidth of 7.5 MHz. Encoding with 4 bits/symbol would increase the susceptibility of the transmitted signal to noise but would only require a bandwidth of 1.875 MHz, as a minimum.

The Common Intermediate Format (CIF) is a common format for transmitting video images at 30 frames/sec over a T1 line (capacity 1.544 Mbits/sec) or a Quarter-CIF (QCIF) picture at 10 frames/sec over Integrated Services Digital Network (ISDN) telephone line (capacity 64 Kbits/sec).

A CIF image is composed of a luminance channel with a resolution of 288 lines/frame x 352 pixels/line and 8 bits/pixel, and two chrominance

channels (C_b and C_r) with half resolution, i.e., 144 lines/frame x 176 pixels/line and 8 bits/pixel. At 30 frames/sec, the data rate is 36.5 Mbits/sec. Transmission over a T1 line for video conferencing applications requires a compression ratio of about 24:1. A QCIF image is one-fourth the size of a CIF image and at 10 frames/sec requires about 3 Mbits/sec. Thus, video phone over ISDN network requires about 48:1 compression.

Advanced Digital Television (ADTV)

The ATDV was developed by the Advanced Television Research Consortium and uses MPEG compression to provide a data rate of 24,000,000 bits/sec. It has two trellis-coded 32-QAM data carriers to provide a wide bandwidth standard priority channel and a narrower bandwidth high-priority channel, all within a 6-MHz transmission bandwidth. The high priority channel provides the viewable picture, and the additional standard-priority channel provides the full HDTV quality. The ADTV spectrum is shown in Figure 7.

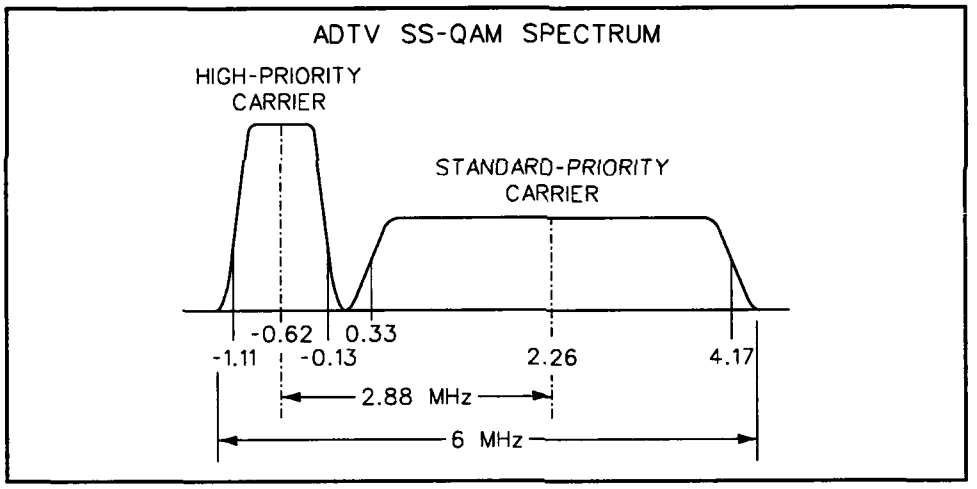

Figure 7. ATDV 32-QAM channel spectrum

The ADTV's 59.94 field rate is identical to that of the National Television Standard Committee (NTSC), thus eliminating temporal artifacts and the need for frame synchronization in mixed ADTV-NTSC environments. Its 1,440x960, 1,050-line scanning format is cost-effective in the production studio. The 2:1 vertical ratio with 525-line NTSC video and 2:1 horizontal ratio with the CCIR Rec. 601 sampling standard used in the 525-line D1 tape recorders offer economical transcoding in mixed ADTV-NTSC production environments. The 16-Mbit DRAM frame memories in an ADTV receiver are predicted to cost about $13 each by 1996.

Digital Spectrum Compatible (DSC)

The DSC was developed by Zenith and AT&T to provide all digital definition television simulcast by compressing the wide bandwidth digital signal into a 6-MHz channel. It uses a unique four-level vestigal sideband (4-VSB or 2 bits/symbol) modulation technique to assure noise free and interference free reception. The 4-VSB coding is complemented by a two-level digital data system (2-VSB, or 1 bit/symbol). The resulting bi-rate coding system identifies and selects the most important picture information on a scene-by-scene basis and automatically transmits that data in a two-level (1 bit/symbol) binary mode. The two-level digital coding makes the system far more tolerant to noise and other interference at greater distances from the transmitter.

The DSC uses a 17-Mbps data rate and a 787.5-line progressive scanning format to eliminate artifacts due to interlacing video.

American Television Alliance Digicipher (ATVA)

The ATVA was developed by General Instrument Corporation and provides another alternative for squeezing a HDTV signal into a 6-MHz bandwidth. Compression is based on the DCT transform. With the ATVA, there are two distinct transmission modes, 32-QAM and 16-QAM. The broadcaster can select the mode, and the receivers can auto-configure to the mode being transmitted. The 32-QAM mode requires 16.5-dB signal-to-noise ratio for error free transmission; whereas, the 16-QAM mode requires only a 12.5 dB signal-to-noise ratio.

Figure 8 shows a block diagram of the encoder. The digital video encoder accepts YUV (Y = luminance, U and V = chromaticities) inputs with 16:9 aspect ratio and 1,050-line interlace (1,050/2:1) at a 59.94 field rate. The YUV signals are obtained from analog RGB input by low-pass filtering, A/D conversion, and an RGB-to-YUV matrix. The sampling frequency is 53.65 MHz for R, G, and B. The digital video encoder implements the compression algorithm and generates a video data stream.

The multiplexer combines the various data streams into one data stream at 18.22 Mbps. The forward error correction (FEC) encoder adds error correction overhead bits and provides 24.39 Mbps of data to the 32-QAM modulation. The symbol rate of the 32-QAM signal is 4.88 MHz.

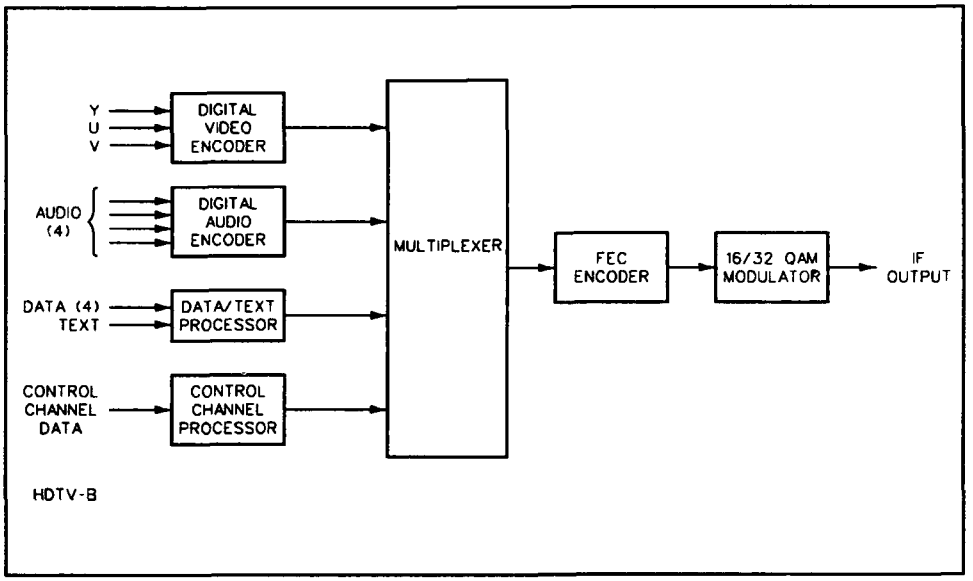

Figure 8. ATVA encoder block diagram

4 Achieving High Compression Ratios

Adaptive Cosine Transforms Using Pointers

The restored image after 25:1 compression in Figure 3 is a result of a simple 8x8 DCT with thresholding and entropy encoding. Typically, Huffman encoding is not done in this report in order to produce a final compressed file; rather, the entropy is calculated according to Equation 1 to determine the maximum amount of compression that can be obtained from Huffman encoding. If the DCT matrices are sparse and contain a large number of zero entries, then Huffman encoding generally produces better results than Arithmetic, or LZW, but usually still falls short of the entropy. The results of several lossless algorithms applied recursively to different files are outlined below.

```
File Name:          Mouse.GS   320x200
Ideal Compression:  1.438:1  (Entropy)
File Size:          64,000 Bytes

          Method:  Huffman 0-Order

Recursion Level        Size   Compression Ratio

        1              45195         1.416
        2              44048         1.453
        3              44306         1.444
        4              44579         1.436
        5              44854         1.427
        6              45129         1.418
        7              45404         1.410
        8              45680         1.401
        9              45956         1.393
```

Method: Adaptive Huffman

Recursion Level	Size	Compression Ratio
1	42617	1.502
2	42527	1.505
3	42850	1.494
4	43163	1.483
5	43476	1.483
6	43789	1.462
7	44110	1.451
8	44431	1.440
9	44751	1.430

Method: LZW

Recursion Level	Size	Compression Ratio
1	56274	1.137
2	77688	0.824
3	76959	0.832
4	89957	0.711
5	103347	0.619
6	116516	0.549
7	133250	0.480
8	151935	0.421
9	173342	0.369

Method: Arithmetic

Recursion Level	Size	Compression Ratio
1	44846	1.427
2	45009	1.422
3	45213	1.416
4	45436	1.409
5	45658	1.402
6	45883	1.395
7	46118	1.388
8	46351	1.381
9	46585	1.374

File Name: Gold.RGB 720x560
Ideal Compression: 1.715:1 (Entropy)
File Size: 403,208 Bytes

Method: Huffman 0-Order

Recursion Level	Size	Compression Ratio
1	238121	1.693
2	220184	1.831
3	219391	1.838
4	219705	1.835
5	220040	1.832
6	220379	1.830
7	220714	1.827
8	221054	1.824
9	221393	1.821

Method: Adaptive Huffman

Recursion Level	Size	Compression Ratio
1	199988	2.016
2	186428	2.163
3	186763	2.159
4	187154	2.154
5	187535	2.150
6	187922	2.146
7	188306	2.141
8	188702	2.137
9	189104	2.132

Method: LZW

Recursion Level	Size	Compression Ratio
1	194810	2.070
2	265701	1.518
3	276317	1.459
4	322433	1.251
5	363944	1.108
6	415781	0.970
7	471263	0.856
8	538827	0.748
9	614199	0.656

Method: Arithmetic

Recursion Level	Size	Compression Ratio
1	236437	1.705
2	236572	1.704
3	236817	1.703
4	237063	1.701
5	237305	1.699
6	237551	1.697
7	237795	1.696
8	238041	1.694
9	238283	1.692

File Name: Lena.D10 512x512
Ideal Compression: 26.7:1 (Entropy)
File Size: 262,144 Bytes

Method: Huffman 0-Order

Recursion Level	Size	Compression Ratio
1	38319	6.841
2	14285	18.351
3	11864	22.096
4	11921	21.990
5	12164	21.551
6	12423	21.102
7	12676	20.680
8	12934	20.268
9	13191	19.873

Method: Adaptive Huffman

Recursion Level	Size	Compression Ratio
1	37821	6.931
2	12993	20.176
3	11382	23.031
4	11546	22.704
5	11837	22.146
6	12135	21.602
7	12440	21.073
8	12745	20.568
9	13035	20.111

Method: LZW

Recursion Level	Size	Compression Ratio
1	11492	22.811
2	15326	17.105
3	16601	15.791
4	19665	13.330
5	22337	11.736
6	25553	10.259
7	29169	8.987
8	33213	7.893
9	38045	6.890

Method: Arithmetic

Recursion Level	Size	Compression Ratio
1	30055	13.416
2	30165	13.367
3	30404	13.262
4	30645	13.157
5	30886	13.055
6	31125	12.954
7	31363	12.856
8	31604	12.758
9	31846	12.661

The source code for the Huffman 0-Order, Adaptive Huffman, Arithmetic coding, and LZW was presented in detail by Nelson (1991). MOUSE.GS and GOLD.RGB represent standard images without much lossless compressibility, while LENA.D10 is a sparse file of DCT coefficients consisting mostly of zeroes. In general, if the entropy calculation is a large number (not much compressibility), then Huffman 0-Order can produce results close to the entropy and dynamic Huffman can exceed the entropy. For a sparse file, the Huffman 0-Order and Adaptive Huffman encoding methods are not very effective unless done at least twice recursively, while the LZW method gives better results on the first recursion. A recursion basically attempts to compress an already compressed file using the same method repeatedly. The fact that none of these methods adequately approaches the entropy for a sparse file indicates that there is room for improvement.

If it were possible to know the location of the zero entries in each 8x8 block, then only the nonzero elements would need to be stored, and the resulting file could be compressed even more tightly than indicated with a $p_i \cdot log_2 p_i$ entropy calculation. The use of overhead bits to indicate the position of the nonzero elements in an 8x8 block of DCT coefficients can

be minimized by taking advantage of the fact that the nonzero entries are clustered around the direct current (DC) component in the upper left-hand corner of the 8x8 matrix.

The least significant bit (LSB) of each nonzero element in the 8x8 matrix is confiscated to use as an indicator bit. This indicator bit tells if there are any more nonzero elements in the remainder of the 8x8 matrix. A "one" in the LSB position is interpreted to mean that the next element contains a nonzero value. A "zero" in the LSB position means that the remaining coefficients are zero. Starting at the upper left-hand corner (DC coefficient), a zig-zag pattern is used to transverse the matrix until a zero is found in the LSB (Figure 9). The resulting file can be Huffman encoded to provide approximately twice as much compression as would be obtained from simple thresholding followed by Huffman encoding (Figure 10).

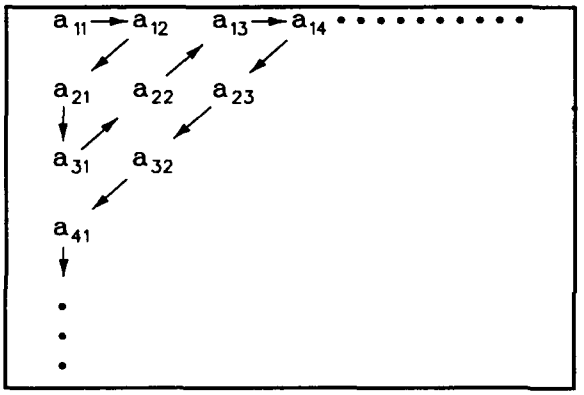

Figure 9. For each element along the zig-zag path, if the remainder of the matrix is zero, then the LSB of that element is zero, otherwise it is set to one

Fourier Interpolation

The Cosine Transform provides a method to increase the resolution of an image by interpolating between known pixels. The Fourier interpolation process (Figure 11) works as follows. To expand an image by 4:1, take the Cosine Transform of the original image in 8x8 blocks, and reconstruct the image using a 16x16 inverse Cosine Transform algorithm. The DCT coefficients in the original 8x8 transform block become the upper left quadrant in the 16x16 DCT block, with the other three quadrants padded with zeroes. The effect of taking the 16x16 inverse transform is equivalent to 2-D sinusoidal interpolation between the pixels in the original image. This process can be generalized to expand an image to any arbitrary size.

Figure 10. Compression and restoration of DBSJ.4 by 19.65:1 by inserting pointers along the zig-zag path with the original image in Figure 3 (MSE = 8.13)

The Relation of Spatial Subsampling to Zonal Filtering

The next logical step toward achieving high compression ratios would be to subsample an image at a rate of 16:1 and take the 8x8 DCT of the resulting subsampled image. If the DCT provided a compression of 6:1, then an overall compression of 96:1 is achieved. The restored image can be magnified by a 16:1 by using the Fourier interpolation technique previously discussed. Figure 12 shows one possible pattern to produce a 16:1 subsampled image prior to taking the DCT.

The scheme shown in Figure 12 does not provide an optimal way of taking the subsample. It is desired to create a 16:1 subsampled image in such a way that the interpolated image is optimally close to the original image. The zonal mask applied to the DCT transform creates an optimal subsampled image.

The process works as follows. Given an image, divide it into 32x32 blocks. Take the DCT of each 32x32 block, but retain only the first 16 elements (4x4) in the upper left-hand corner. The other 1,008 elements are forced to zero.

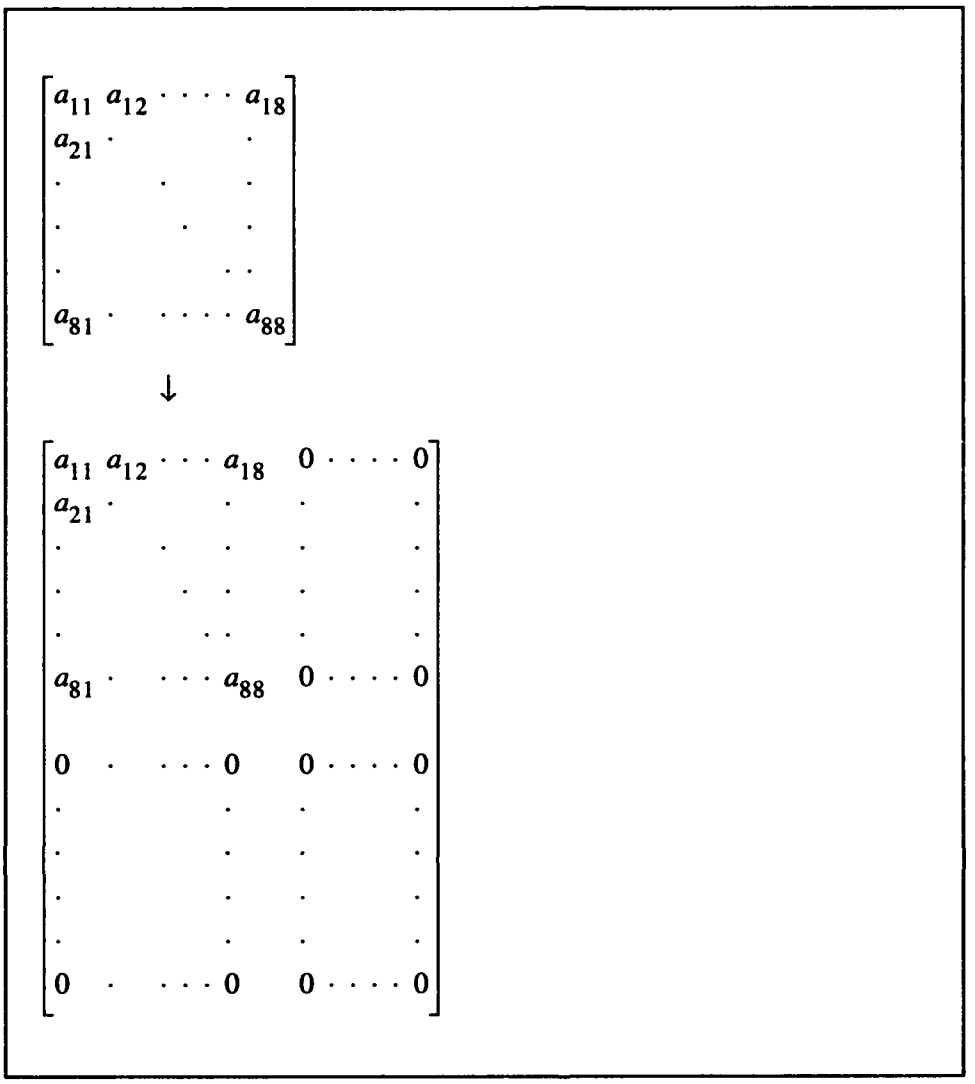

Figure 11. Fourier interpolation of an 8x8 transformed image by taking the inverse transform in 16x16 blocks with three quadrants padded with zeroes

If an inverse 4x4 DCT is performed on the remaining elements, then an optimal subsampled image is produced. In this case, the subsampling ratio is 64:1. Huffman encoding of the 4x4 DCT blocks will normally result in overall compression ratios of 80:1 to 160:1.

The restoration process includes:

a. Huffman decoding to recover the coefficients of each 4x4 transform block.

b. Creating 32x32 blocks from the 4x4 transform coefficients by padding the remaining 1,008 elements with zero.

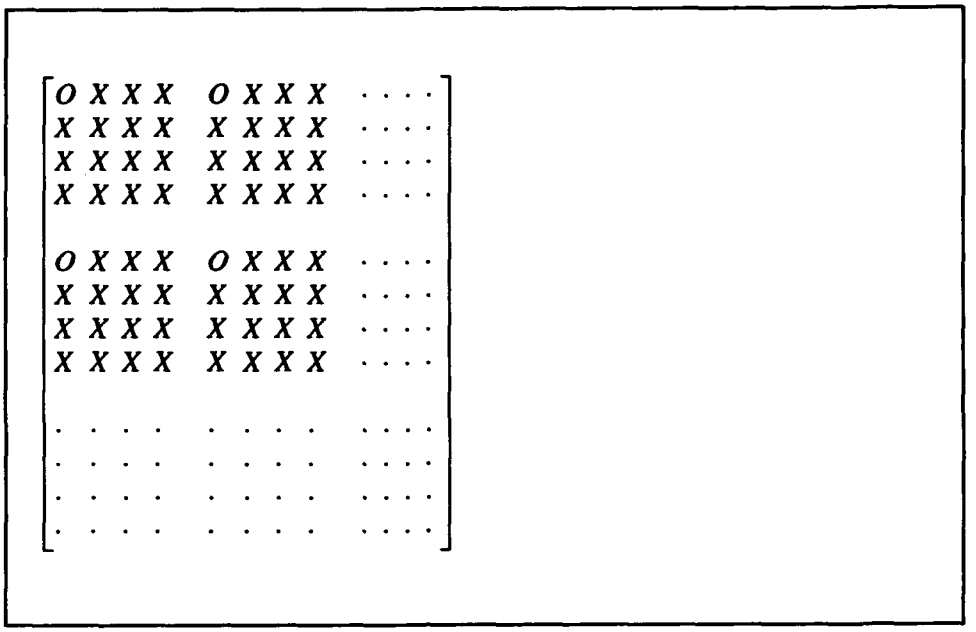

Figure 12. The original image consists of those pixels shown with X's and O's, but only the O's are used to a 16:1 subsampled image

c. Taking the inverse 32x32 discrete cosine transform.

This process is optimal in the following sense:

a. The discrete cosine transform is a near optimal transform in the mean-square sense and produces results equivalent to the Karhunen-Loeve transform for images that have a high degree of adjacent pixel correlation.

b. By utilizing the transform of the original image over 32x32 blocks, instead of 8x8 blocks, an advantage is taken of any correlation of pixels that may exist between adjacent 8x8 blocks. The JPEG algorithm cannot achieve compression rates as high because its block size is initially limited to 8x8.

c. A 4x4 zonal mask applied to the 32x32 transform creates an optimal subsample so that the interpolation process on reconstruction will be as close as possible to the original image.

A list of test images is shown below. Figures 13 through 17 display the original image beside the reconstructed image for comparison. In these figures, both the original and the reconstructed image have been scaled down to allow them to be placed side-by-side on a single page for comparison. In Figure 18, the left image is a zoom-in of the original PETRA; the right image is a zoom-in of the reconstructed image. This display is a true pixel-by-pixel comparison.

Image Name	Actual Size	Displayed Size
Petra	1,636x2,152	320x340
Vale	2,523x1,617	504x323
Twins	2,529x1,578	505x315
Room	2,110x2,695	422x539
Turbans	2,523x1,617	504x323

Figure 13. Petra - original image (left) and compressed 91.91:1 and restored (right)

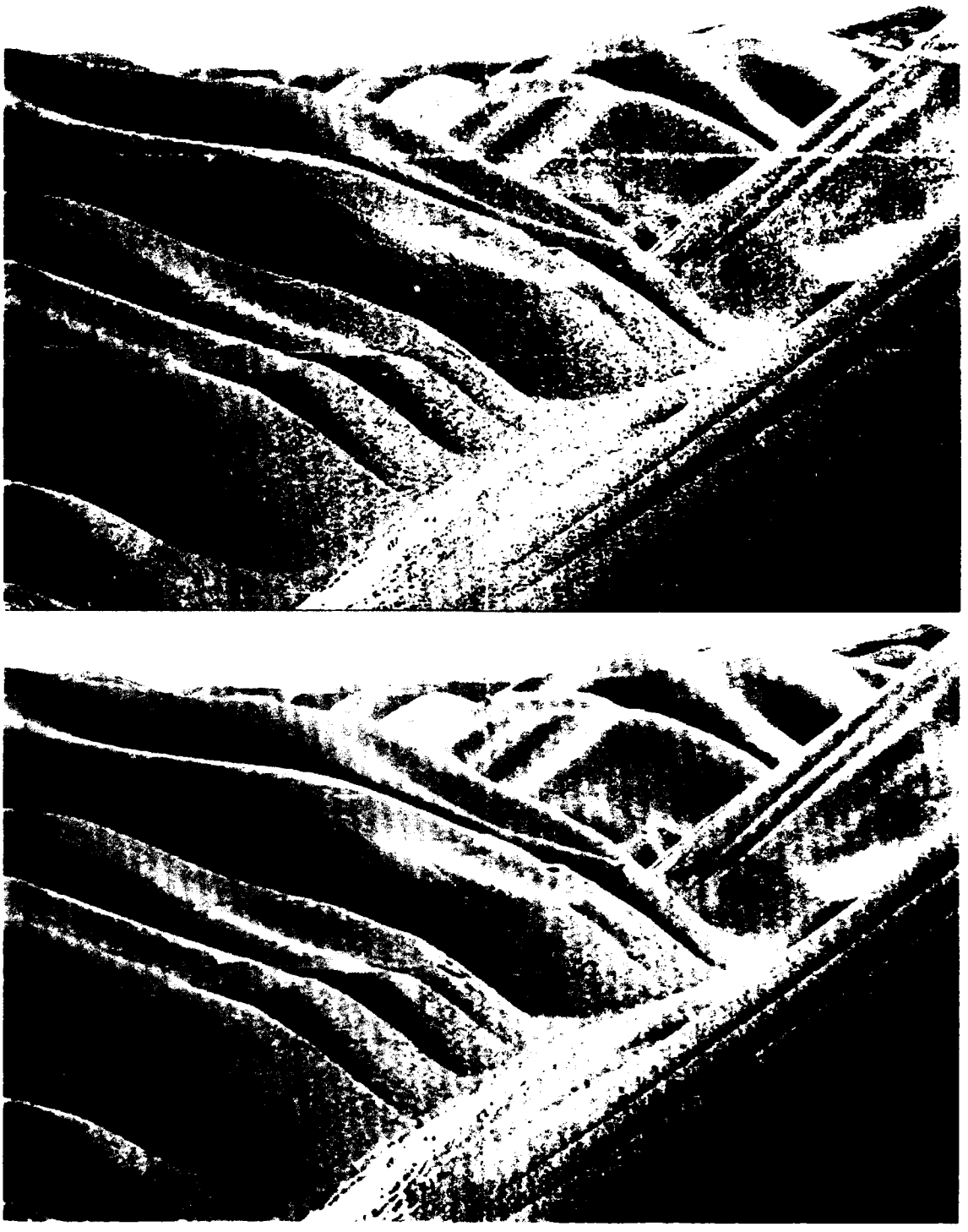

Figure 14. Vale - original image (top) and compressed 120.97:1 and restored (bottom)

Figure 15. Twins - original image (top) and compressed 89.31:1 and restored (bottom)

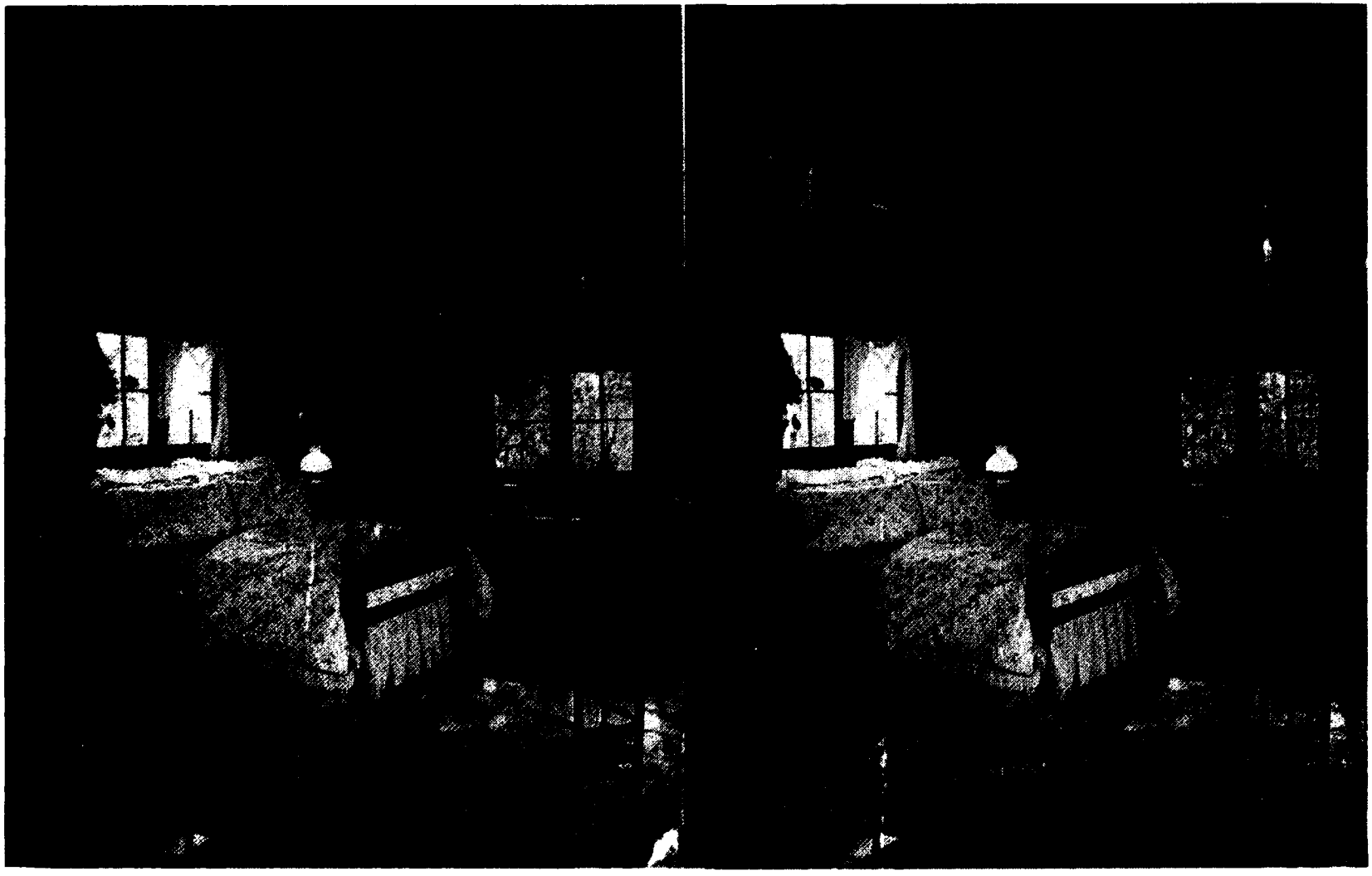

Figure 16. Room - compressed 159:1 and restored image (left) and original (right)

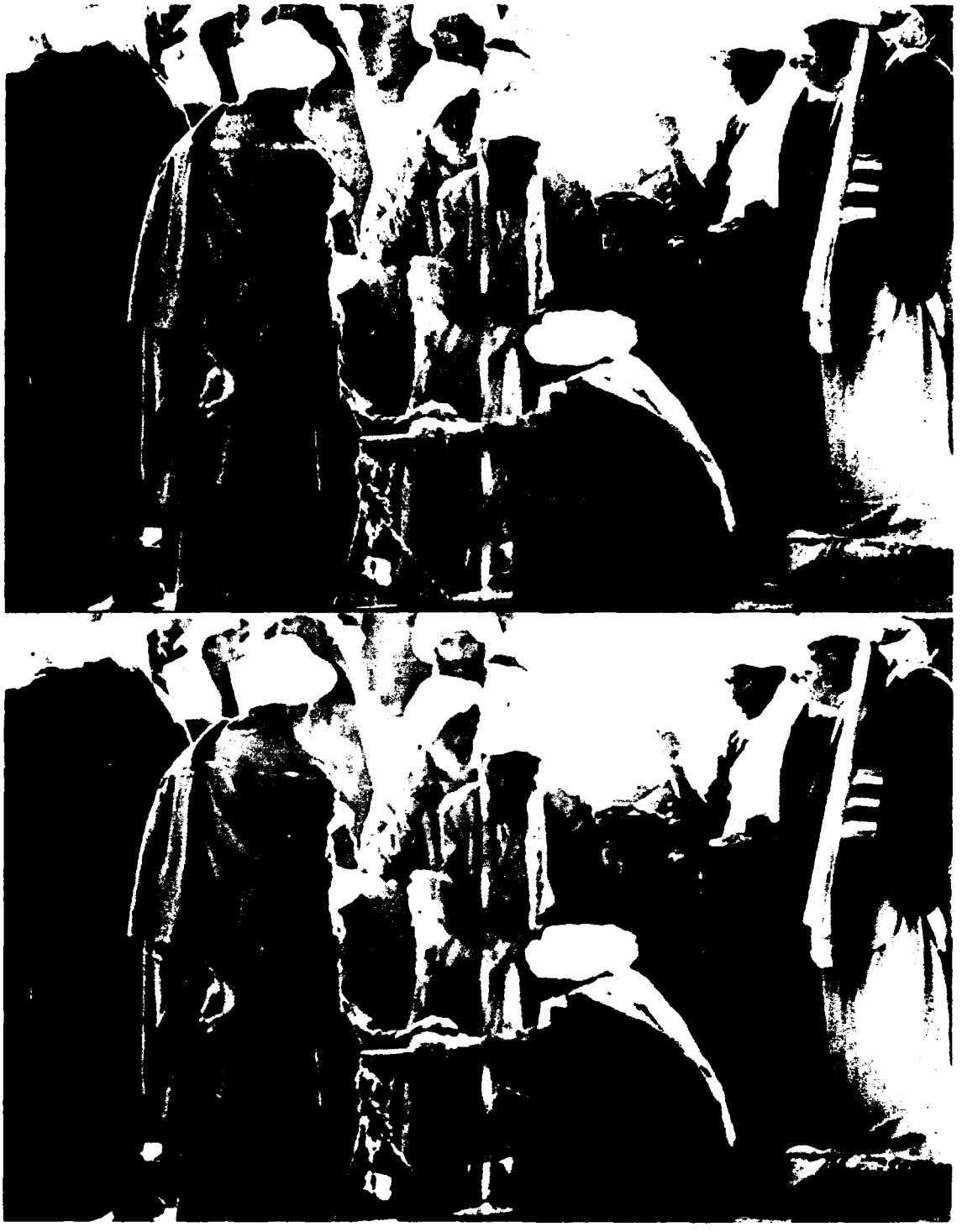

Figure 17. Turbans - original image (top) and compressed 100.9:1 and restored (bottom)

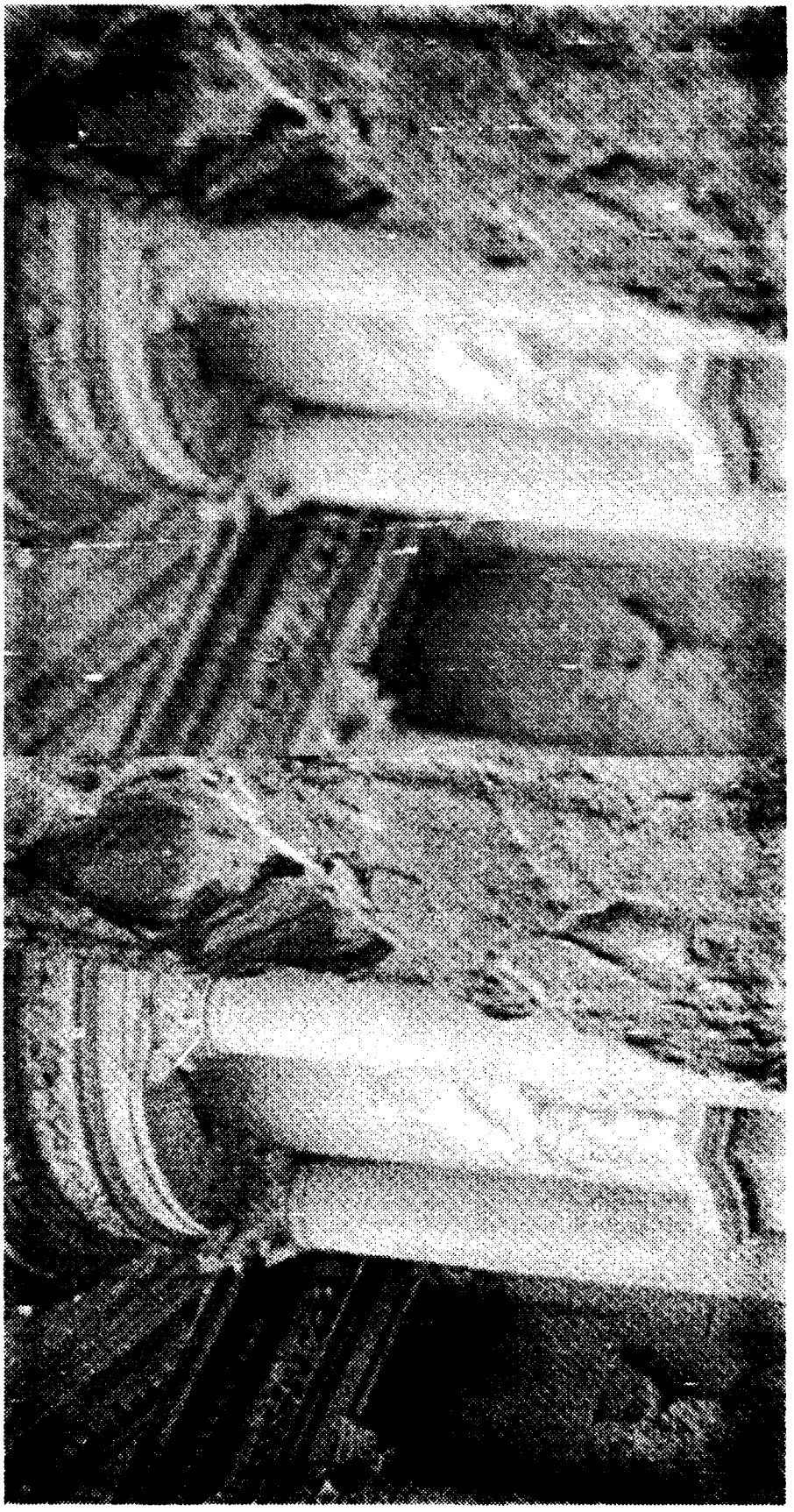

Figure 18. Zoom-in of the original PETRA (left); zoom-in of reconstucted image (right)

References

Barnsley, M. (1988). *Fractals everywhere,* Academic Press, Inc., Norcross, GA.

Couch, L. (1983). *Digital and analog communication systems,* Macmillan Publishing Co., New York.

Jain, A. K. (1989). *Fundamentals of digital image processing,* Prentice-Hall, Englewood Cliffs, NJ.

Nelson, M. (1991). *The data compression book,* M&T Books, Redwood City, CA.

"Software listings," *Dr. Dobb's Journal.* (1991). M&T Books, Redwood City, CA.

Bibliography

Barnsley, M. (1992). "Methods and apparatus for image compression by iterated function system," U.S. Patent 4,941,193.

Ferraro, R. F. (1990). *Programmer's guide to the EGA and VGA cards,* 2nd ed., Addison-Wesley, Reading, MA.

Jacob, Lempel, A., and Ziv, J. (1977). "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory.*

_____ . (1978). "Compression of individual sequences via variable-rate coding," *IEEE Transactions of Information Theory.*

"NCSA image 3.1," National Center for Supercomputing Applications, University of Illinois, Urbana-Champaign.

"386-MATLAB user's guide." (1990). The Mathworks, Inc. Natick, MA.

Welch, T. (1984). "A technique for high-performance data compression," *IEEE Computer,* 17(6), 8-19.

Appendix A
Compressions with the Cosine Transform and Singular Value Decomposition (SVD)

The following set of images demonstrates increasing distortion because of increasing compression rates for two methods - the Cosine Transform and SVD. This display is by no means a comparison of the two methods. The Cosine Transform is performed in 8x8 blocks; whereas, SVD is performed over the entire image. If SVD were executed in 8x8 blocks, the method's ratio of compression to distortion would most likely improve.

Figure A1. Original image

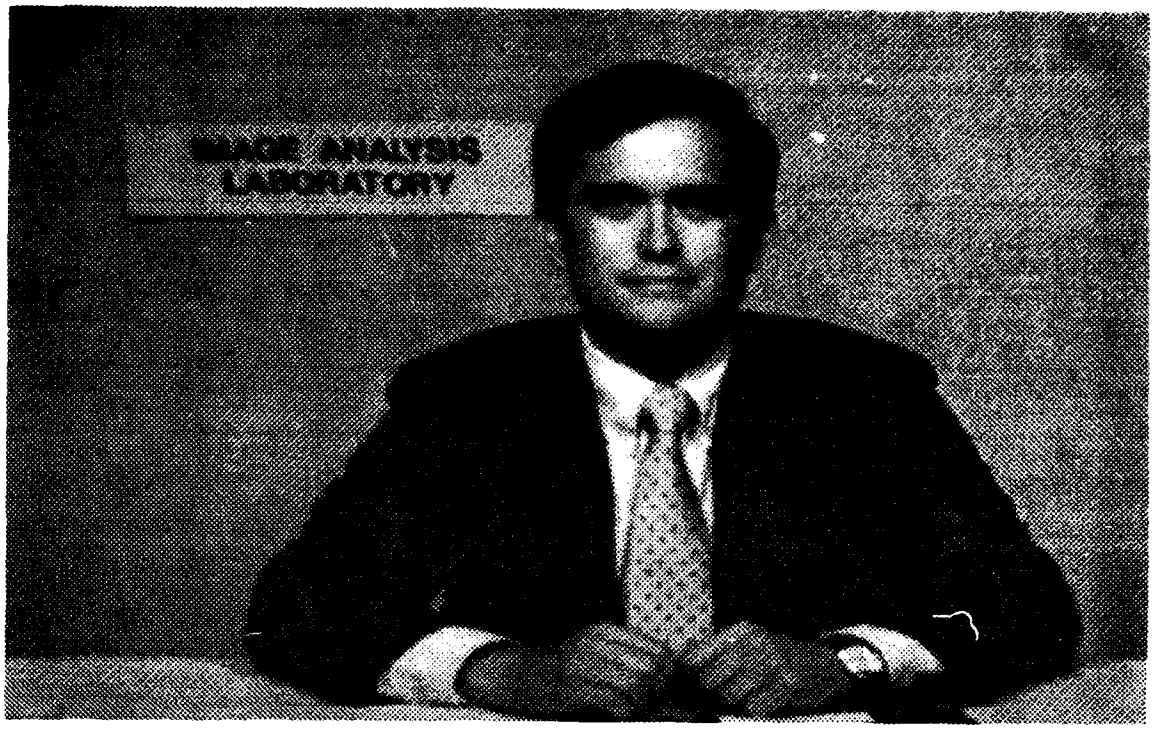

Figure A2. DCT encoding followed by Huffman encoding then Huffman decoding and inverse DCT (MSE = 2.734; compression with Huffman encoding of DCT coefficients = 7.186:1)

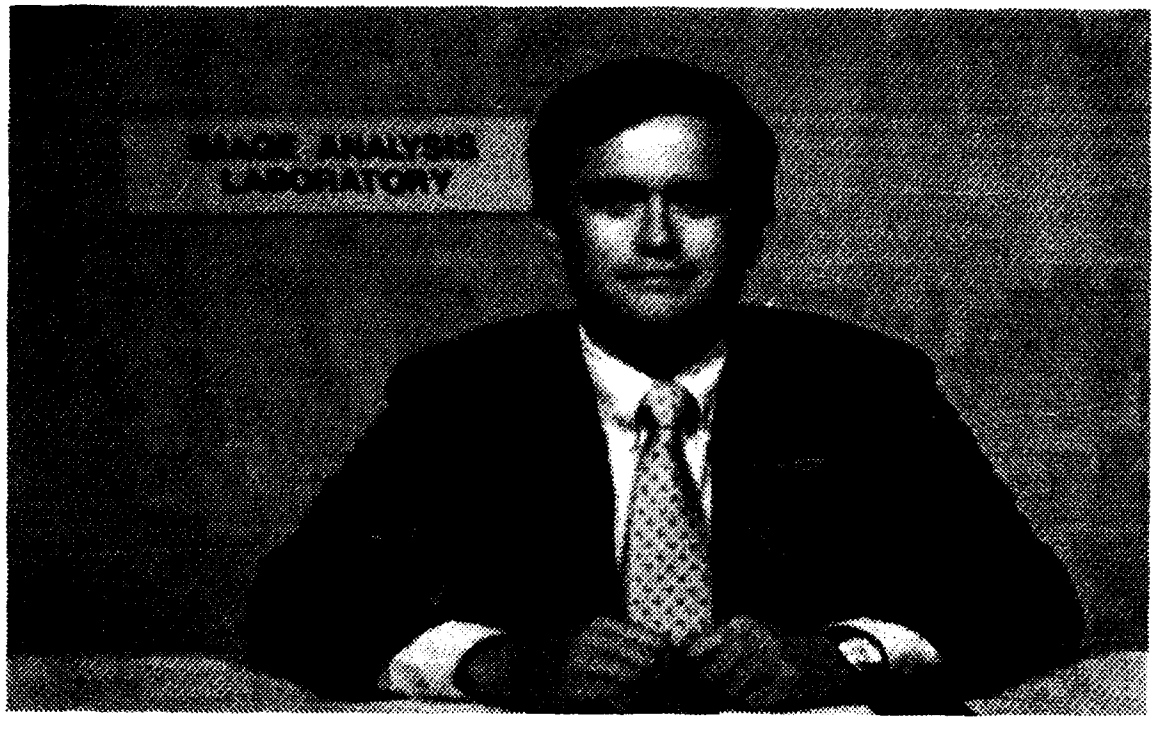

Figure A3. DCT encoding with a threshold of 1 followed by Huffman encoding, then Huffman decoding and inverse DCT (MSE = 5.859; compression with Huffman encoding of DCT coefficients = 12.326:1)

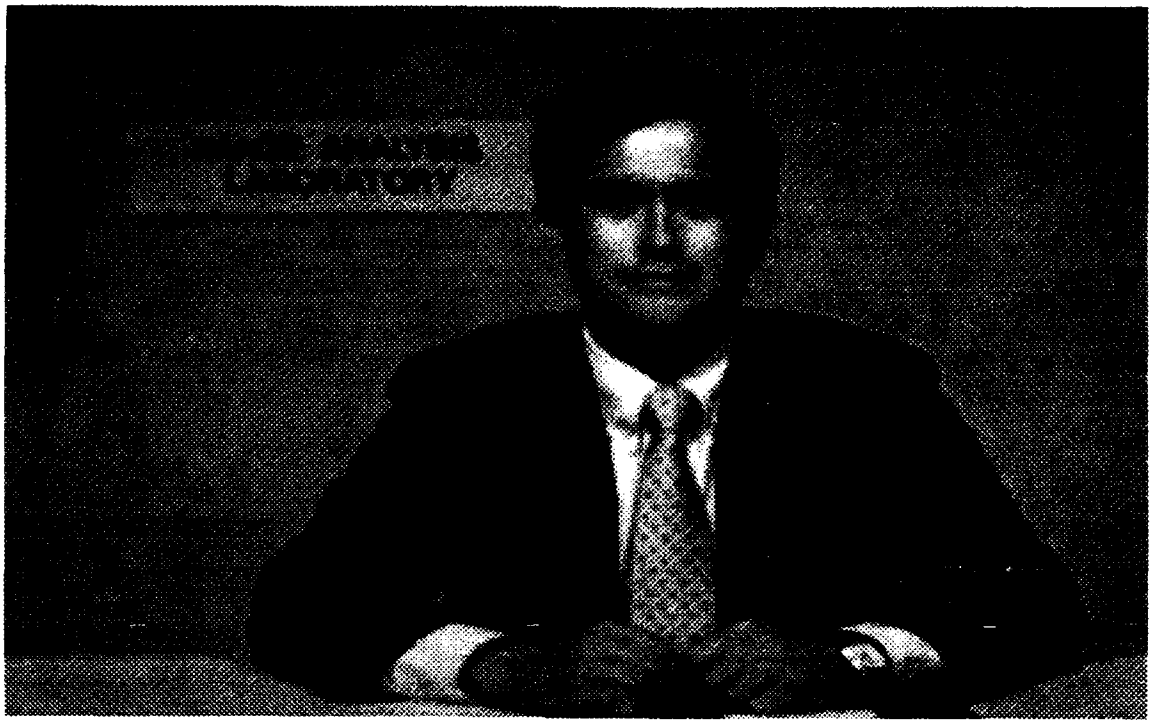

Figure A4. DCT encoding with a threshold of 2 followed by Huffman encoding, then Huffman decoding and inverse DCT (MSE = 9.818; compression with Huffman encoding of DCT coefficients = 15.659:1)

Figure A5. DCT encoding with a threshold of 3 followed by Huffman encoding, then Huffman decoding and inverse DCT (MSE = 14.544; compression with Huffman encoding of DCT coefficients = 18.488:1)

Figure A6. DCT encoding with a threshold of 4 followed by Huffman encoding, then Huffman decoding and inverse DCT (MSE = 20.047; compression with Huffman encoding of DCT coefficients = 21.13:1)

Figure A7. DCT encoding with a threshold of 5 followed by Huffman encoding, then Huffman decoding and inverse DCT (MSE = 25.39; compression with Huffman encoding of DCT coefficients = 23.34:1)

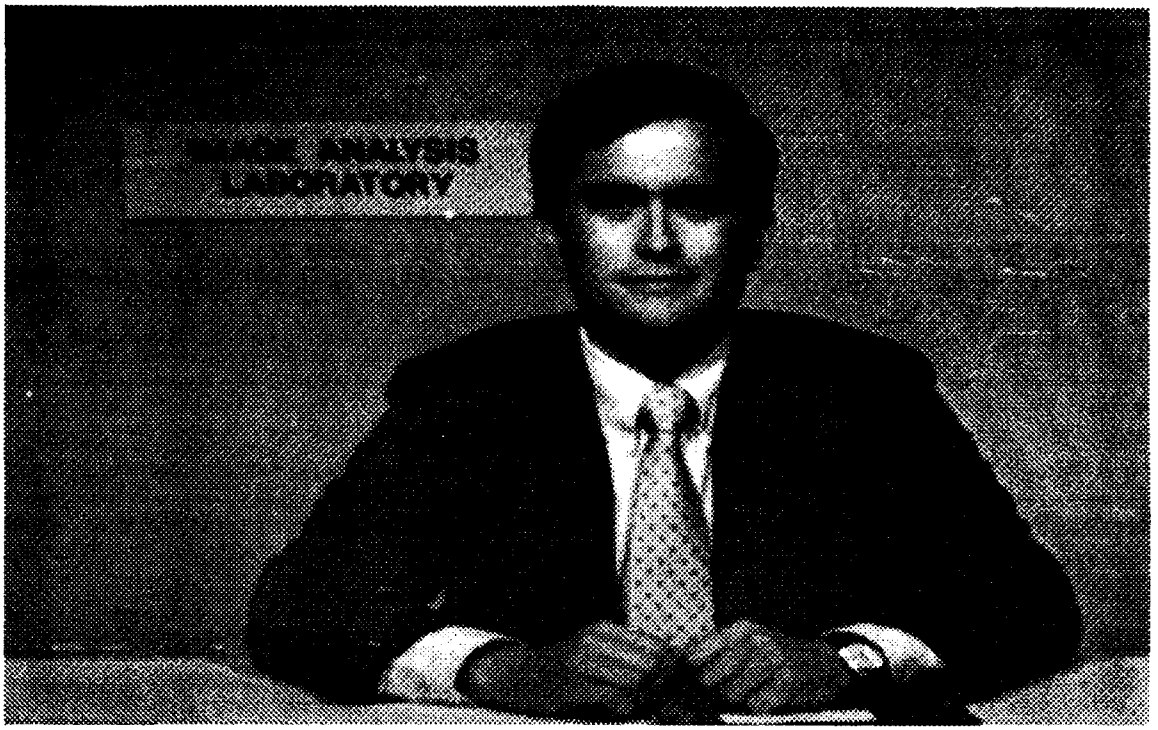

Figure A8. A 2:1 compression with SVD (MSE = 0.765)

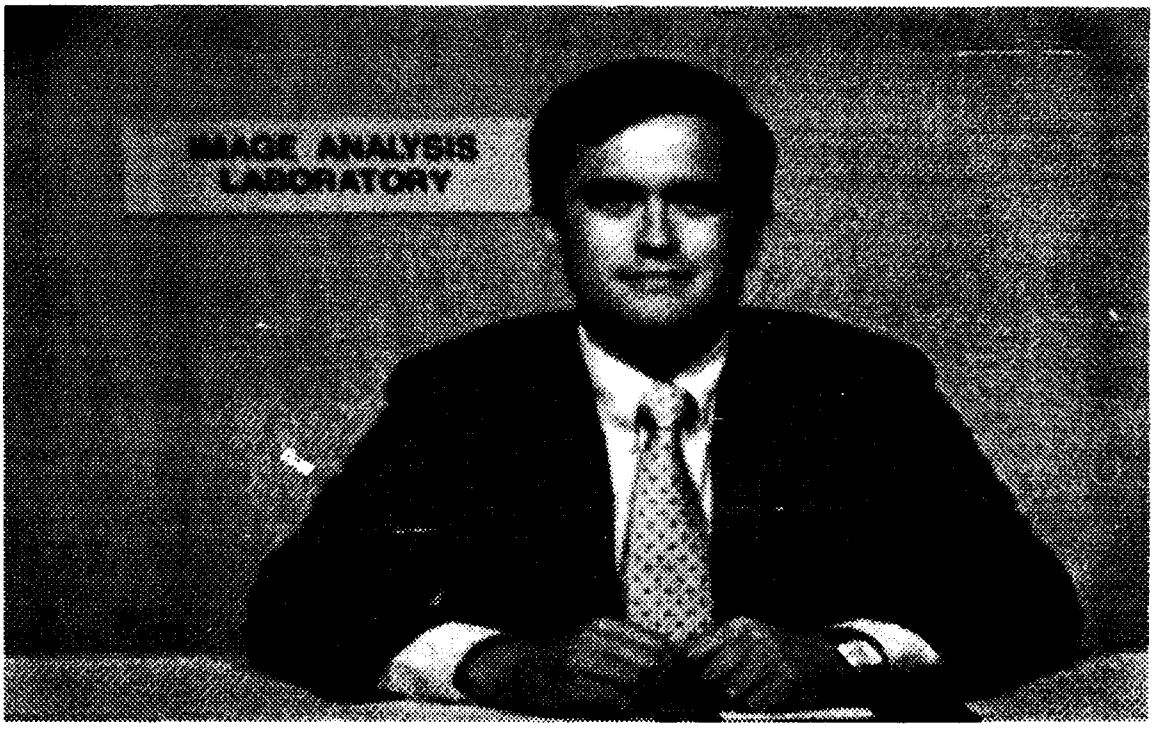

Figure A9. A 4:1 compression with SVD (MSE = 3.802)

Figure A10. A 10:1 compression with SVD (MSE = 29.584)

Figure A11. A 25:1 compression with SVD (MSE = 124.2)

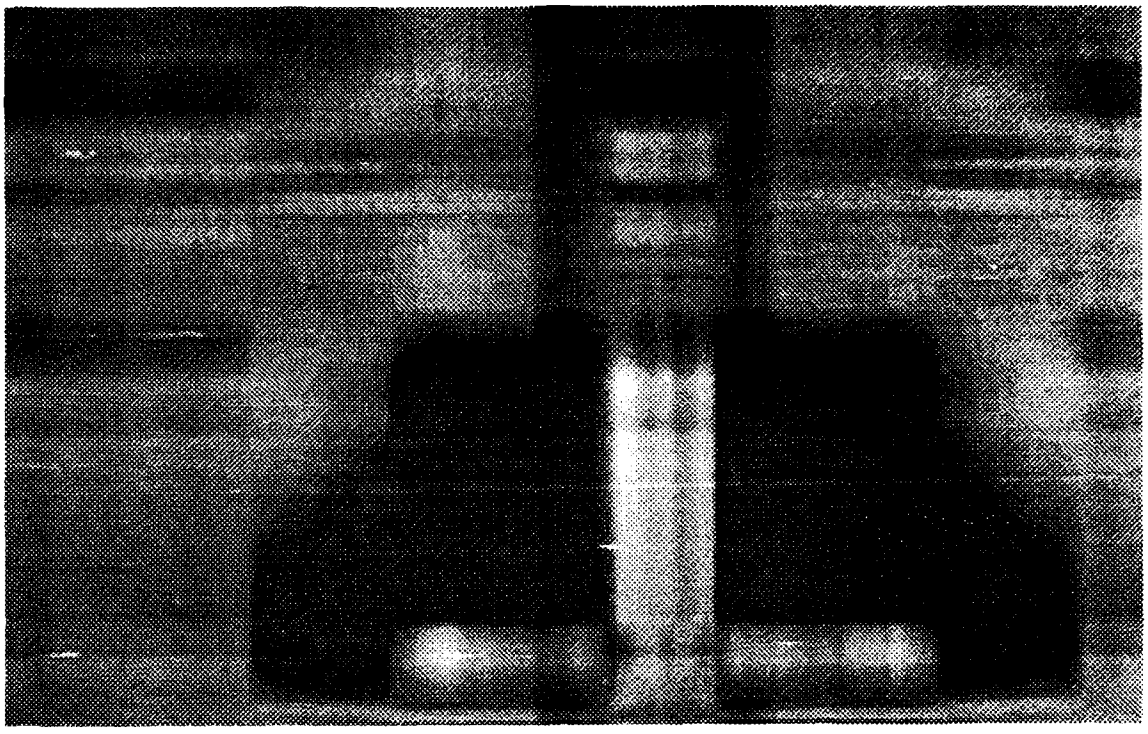

Figure A12. A 100:1 compression with SVD (MSE = 552.95)

Appendix B
"Image Lab" Software User's Guide

File

Copy File, Delete File, and **Rename File** perform similarly to their Disk Operating System (DOS) equivalents. **Copy File** copies file information from one file to another. **Delete File** deletes a file. **Rename File** moves file information from one file to another.

Load File requests a image file name in either the RGB format or the greyscale format. Image width and height are requested. This selection is a prerequisite to displaying an image. Once an image file is loaded, it does not need to be reloaded; the most recent image file and its dimensions are stored in global memory. For example, if a different display mode is desired, change the mode in "Options," and then select **Display Full Image** under "View"; the image will then be redisplayed without requiring the user to reload the image information.

Load Palette requests an RGB or greyscale palette name. An RGB palette must list the 256 intensities for (a) red, (b) green, and (c) blue for a total of 768 entries. A RGB pixel is translated by the following mask: R R R G G G B B. Bits 5-7 indicate the intensity of red, bit 7 being the MSB. Bits 2-4 indicate the intensity of green, bit 4 being the MSB. Bits 0-1 indicate the intensity of blue, bit 1 being the MSB. A greyscale palette must list the 256 grey intensities and repeat the list twice for a total of 768 entries. (Note: To obtain any shade of grey, the intensities of red, green, and blue must be equal.) All palettes must be 768 bytes in size. No stray bits are allowed! Once a palette is loaded, it remains in global memory; therefore, the palette does not require reloading from one image to the next. The default palette is GRAYTEST.PAL.

Quit ends the use of "Image Lab."

View

Show Red shows only the red intensities. **Show Green** shows only the green intensities. **Show Blue** shows only the blue intensities. **Show Bit Plane 1-7** masks out all bits except the requested one. If the bit is present in a pixel, white will be displayed, else black will be displayed. **Display Full Image** redisplays the original image.

Analysis

Calculate Entropy calculates the entropy and predicts the theoretical compression ratio for an image or processed file. The theoretical compression ratio is based on Huffman-0. **Chop Image File** allows a subimage to be created from an image file. The original file name and dimensions are requested along with the output file name, starting and ending horizontal coordinates, and the starting and ending vertical coordinates. The top left pixel in the original file is located at X = 1, Y = 1. The starting and ending bounds are *inclusive*. **Compare Files** compares the pixels of two identically sized files. The threshold is the lowest difference that is to be flagged. The output will display four aspects of the differing pixels. Pixel 1 is the value of the pixel in the first file. Pixel 2 is the value of the pixel in the second file. X is the horizontal coordinate. Y is the vertical coordinate. The top left pixel is located at X = 0, Y = 0. This selection is helpful in analyzing images processed by lossy techniques. The original and reconstructed images can be compared on a pixel-by-pixel basis. **Compression Ratio** tells the original file size, the compressed file size, and the ratio. **Difference Images** creates a difference image between original and reconstructed images. Zero error is represented by gray or intensity 128. Increasing positive error is represented by increasing intensity, and increasing negative error is represented by decreasing intensity. **Histogram** makes a text histogram of the pixels in an file. **Mean Squared Error** finds the MSE between two image files. The two files do not have to be the same size. For color processed images, this selection gives distorted results since the color procedures work with RED bits, GREEN bits, and BLUE bits instead of *full* 8-bit pixels. **Paste Image Files: Horizontally** pastes two images side by side; **Paste Image Files: Vertically** pastes one image above the other. The two images do not have to be the same height or width. This selection is helpful in visually comparing original and reconstructed images.

Lossless

Four options are available for compression and decompression: (a) **Huffman-0**, (b) **Adaptive Huffman**, (c) **LZW**, and (d) **Arithmetic**. If a

file is compressed in one option, it must be decompressed in the same option.

Lossy

Three transforms and inverse transforms are available for compression and decompression: (a) **Cosine**, (b) **Hadamard**, and (c) **Sine**. If a file is compressed in one option, it must be decompressed in the same option. **Black & White** processes the image as a greyscale image. For compression, the reconstructible DCT will be located in *.1*; the displayable DCT will be located in the output file specified. For decompression, an *.1* file must exist to reconstruct, or an *.d* file must exist assuming that the image can be cut evenly into blocks specified by the lossy block mode being used. **Color** processes the image as an RGB image by translating to the *Y*, C_b, C_r format as shown in Equations B1-B3.

$$Y = 0.299 * R + 0.587 * G + 0.114 * B \quad \text{(B1)}$$

$$C_b = -0.16874 * R - 0.33126 * G + 0.5 * B \quad \text{(B2)}$$

$$C_r = 0.5 * R - 0.41869 * G - 0.08131 * B \quad \text{(B3)}$$

RGB format is regained with Equations B4-B6.

$$R = Y + 1.402 * C_r \quad \text{(B4)}$$

$$G = Y - 0.34414 * C_b - 0.71414 * C_r \quad \text{(B5)}$$

$$B = Y + 1.772 * C_b \quad \text{(B6)}$$

For compression, the displayable DCT's will be located in *.*yyy*, *.*ccb*, and *.*ccr*; the reconstructible DCT's will be located in *.*1yy*, *.*1cb*, and *.*1cr*. For decompression, *.*1yy*, *.*1cb*, and *.*1cr* files must exist to reconstruct. File names are not a problem as long as all processing is done by this software!

In compression, the DCT is manipulated by two factors: thresholding and zonal filtering. If the absolute value of a DCT entry is less than or equal to the threshold, that entry is set to zero. Zonal filtering is more complicated. The zonal filter level tells what coefficients to keep in each DCT square. For example, if the lossy mode is set to 8x8 and the zonal filter level equals 4, each 8x8 block of the DCT will retain the 16 entries in the top left corner that make a 4x4 square. All coefficients outside the 4x4 are set to zero. To disable zonal filtering, select level 8 for 8x8 mode, 16 for 16x16 mode, and 32 for 32x32 mode.

Options

Set Video Mode displays the possible video modes and allows one to be selected. The default mode is set to the highest resolution that the software can find. If the PC's graphics card is not a Super VGA card, most likely, the only mode that will work is 320x200. If images will not display, set the video mode to 320x200.

Set Lossy Mode allows three different lossy modes to be selected: 8x8, 16x16, and 32x32. Each mode represents the block size that the image will be cut into to be processed. The image file does not have to be cut evenly into squares of the selected block size. For example, a 17x17 image can be processed in any of the lossy modes. The 8x8 and 16x16 modes can process image files with a maximum width of 1,024 pixels. The 32x32 mode is restricted to a maximum width of 640 pixels. The height is boundless. The 8x8 processes files in one pass. If the image width is less than or equal to 512 pixels, the 16x16 processes files in one pass, else it processes files in two passes. The 32x32 requires a pass for each 160-pixel-wide strip. The last strip is not necessarily 160 pixels wide. For example, an image that is 350 pixels wide would require two 160-pixel-wide strips and a third 30-pixel-wide strip. All strips are pasted together at the end of the procedure. As the files are processed, dots are displayed. Each continuous row of dots represents a strip. Lossy color procedures process Y strips first, C_b strips next, and C_r strips last. The default lossy mode is 8x8.

Set Menu Color provides four menu color options: (a) Red, (b) Green, (c) Blue, and (d) Black and White.

Make Palette allows the user to create an RGB palette. The red, green, and blue intensities should be entered in ascending order. Once entered, the intensities are scrambled into an RGB palette.

Appendix C
"Image View" Software User's Guide

File

Load Series File uses information from a file to display a sequential set of images. The file format is as follows:

```
number_of_images  mode1  palette_name1  image_name1  x1  y1  mode2 ...
```

The modes are abbreviated as follows: 1=320x200, 2=640x480, 3=800x600, 4=1024x768. This file must be an ASCII file; if WP is used to generate this file, use CNTL F5 to save the file as ASCII text. Any combination of RGB and/or grayscale images may be displayed. Once a series is loaded, it is stored in global memory; therefore, the series does not need to be reloaded to perform "View" selections.

Quit ends the use of "Image View."

View

Show Red shows only the red intensities. **Show Green** shows only the green intensities. **Show Blue** shows only the blue intensities. **Show Bit Plane 1-7** masks out all bits except the requested one. If the bit is present in a pixel, white will be displayed, else black will be displayed. **Display Full Image** redisplays the original series.

Options

Set Menu Color provides four menu color options: (a) Red, (b) Green, (c) Blue, and (d) Black and White.

Appendix D
Source Code for Individual Programs

ENTROPY.BAS

```
10 DIM J(257), K1(257)
20 CLASS
30 SUM = 0
40 ICOUNT = 0
50 INPUT "Type name for input file or <RET> "; F$
60 PRINT
70 OPEN F$ FOR BINARY AS #1
80 A$ = INPUT$(1, 1)
90 IF EOF(1) THEN GOTO 130
100 ICOUNT = ICOUNT + 1
110 J(ASC(A$)) = J(ASC(A$)) + 1
120 GOTO 80
130 FOR I = 0 TO 255
140 IF J(I) = 0 THEN GOTO 160
150 SUM = SUM - (J(I) / ICOUNT) / LOG(2) * LOG(J(I) / ICOUNT)
160 NEXT I
170 PRINT "The entropy = "; SUM; " bits per symbol."
180 PRINT "A lossless compression of "; 8 / SUM; " to 1 is possible
with entropy coding."
190 CLOSE #1
200 PRINT
210 PRINT "Type any key to end "
220 A$ = INKEY$
230 IF A$ = "" THEN GOTO 220
240 END
```

MSE.BAS

```
10 REM  Program Mean-Square-Error
20 REM this program computes the mean square error between two
```

```
25 REM files
30 n = 1
40 CLASS
50 INPUT "Type filename #1 "; F$
60 INPUT "Type filename #2 "; G$
70 OPEN F$ FOR BINARY AS #1
80 OPEN G$ FOR BINARY AS #2
90 A$ = INPUT$(1, 1)
100 B$ = INPUT$(1, 2)
110 IF EOF(1) THEN GOTO 1000
120 SUM = SUM + (ASC(A$) - ASC(B$)) ^ 2
130 n = n + 1
140 GOTO 90
1000 SUM = SUM / n
1010 PRINT "The mean square error = "; SUM
```

ENTROPY.C

```
#include <math.h>
#include <stdio.h>
/*Huffman Estimator*/
/* Mike Ellis */
main ()
{
char string[80];
unsigned char p;
int i,k;
float j[256],pixel;
float entropy;
FILE *input_file;
FILE *infile;
          entropy = 0;        /*Set all Variable to 0 */
          for ( i = 0; i<=255; i++)
          {
          j[i]=0;
          }
          pixel = 0;
          printf ("\nType name of file ");  /*Open the input file*/
         scanf ("%s",string );
          input_file=fopen(string,"r+b");   /*For Read + Binary */
          while (!feof(input_file))         /*Read to end-of-file*/
         {
          fscanf (input_file,"%c",&p); /*input as unsigned char*/
          pixel = pixel + 1;           /*count characters read*/
          k=p;                         /*convert char to integer*/
          j[k]=j[k]+1;                 /*number of times that */
          }                            /*this char was read */
          for (i = 0; i <=255; i++)    /*compute entropy*/
          {
          if (j[i] !=0)
          {
          entropy = entropy + j[i]*log(j[i]/pixel);
          }
          }
          entropy = -entropy/(pixel*log(2));
          printf ("\nEntropy = %f",entropy); /*print the entropy*/
          printf("\nEntropy encoding can achieve");
          printf(" a %f to 1 compression.\n",8/entropy);
}
```

IMAGE.BAS

```
10 REM
20 REM PROGRAM "IMAGE" FOR DISPLAY GRAYSCALE OR COLOR IMAGES
30 REM USE UNDER QUICKBASIC VERSION 4.5
40 REM
50 CLASS
60 INPUT "Type Image Filename "; C$
70 DEFINT A-Z
80 INPUT "Type X Dimension "; XDIM
90 INPUT "Type Y Dimension "; YDIM
100 GOSUB 210                          'set up 256 color palette
110 OPEN C$ FOR BINARY AS #1
120 WINDOW SCREEN (0, 0)-(XDIM, YDIM)
130 FOR I = 1 TO YDIM
140 A$ = INPUT$(XDIM, 1)
150 FOR J = 1 TO XDIM
160 B = ASC(MID$(A$, J, 1))
170 PSET (J, I),                       'display the pixel
180 NEXT
190 NEXT
200 GOTO 200
210 INPUT "Type Palette Filename "; P$
220 SCREEN 13
230 OPEN P$ FOR BINARY AS #1
240 DIM RED(256), GREEN(256), BLUE(256)
250 FOR K = 0 TO 255
260 A$ = INPUT$(1, 1)
270 RED(K) = ASC(A$)
280 NEXT K
290 FOR K = 0 TO 255
300 A$ = INPUT$(1, 1)
310 GREEN(K) = ASC(A$)
320 NEXT K
330 FOR K = 0 TO 255
340 A$ = INPUT$(1, 1)
350 BLUE(K) = ASC(A$)
360 NEXT K
370 FOR I = 0 TO 255
380 PALETTE I, (INT(BLUE(I) / 4)) * 65536! + 256 * (INT(GREEN(I)/
4)) + INT(RED(I)) / 4
390 NEXT I
400 CLOSE #1
410 RETURN
```

```
10 COMMON DATA1%()                          'More than 64K block
20 CLEAR
30 DIM C(8, 8), DATA1%(90, 1, 8, 8), TEMP(8, 8)
40 DIM TEMP2(8, 8)
50 MAX(1, 1) = -100000!
60 MIN(1, 1) = 100000!
70 REM
80 REM Collect Input Data 8 X 8 Byte Blocks
90 REM
100 INPUT "Type Name of Input File "; F$
110 IF F$ = "" THEN FILES: GOTO 100
120 K = INSTR(1, F$, ".")
130 IF K <> 0 THEN A$ = LEFT$(F$, K - 1) ELSE A$ = F$
140 INPUT "Type X Range "; XINDEX
150 INPUT "Type Y Range "; YINDEX
160 XINDEX = (XINDEX / 8)
170 YINDEX = (YINDEX / 8)
180 K1 = XINDEX * YINDEX
190 G$ = A$ + ".dct"
200 OPEN F$ FOR BINARY AS #1
210 OPEN G$ FOR OUTPUT AS #2
260 GOSUB 870                     'Define cosine transform matrix
290 REM
300 FOR J = 1 TO YINDEX
310 PRINT J * 8
320 FOR Y = 1 TO 8
330 I1 = 0
340 A1$ = INPUT$(8 * XINDEX, 1)
350 FOR I = 1 TO XINDEX
360 FOR X = 1 TO 8
370 I1 = I1 + 1
380 A$ = MID$(A1$, I1, 1)
390 DATA1%(I, 1, Y, X) = ASC(A$) - 128
400 NEXT X
410 NEXT I
420 NEXT Y
430 GOSUB 590                      'Perform the cosine transform
440 FOR X = 1 TO 8
450 FOR I = 1 TO XINDEX
460 FOR Y = 1 TO 8
470 K = (DATA1%(I, 1, X, Y))
480 S$ = S$ + CHR$(K + 128)
490 NEXT
500 NEXT
510 PRINT #2, S$;
520 S$ = ""
530 NEXT
540 REM
550 NEXT
```

```
560 CLOSE #1
570 CLOSE #2
580 END
590 REM
600 REM
610 FOR I = 1 TO XINDEX
620 REM FOR k = 1 TO 8
630 FOR L = 1 TO 8
640 TEMP1 = 0
650 FOR M = 1 TO 8
660 FOR N = 1 TO 8
670 TEMP1 = TEMP1 + DATA1%(I, 1, M, N) * C(L, N)
680 NEXT
690 TEMP(M, L) = TEMP1
700 TEMP1 = 0
710 NEXT
720 NEXT
730 REM NEXT
740 FOR L = 1 TO 8
750 TEMP1 = 0
760 FOR M = 1 TO 8
770 FOR N = 1 TO 8
780 TEMP1 = TEMP1 + C(M, N) * TEMP(N. L)
790 NEXT N
800 DATA1%(I, 1, M, L) = TEMP1 / 8
810 TEMP1 = 0
820 NEXT
830 NEXT
840 REM
850 NEXT I
860 RETURN
870 REM
880 REM Generate Cosine Transform Matrix
890 FOR K = 0 TO 7
900 FOR N = 0 TO 7
910 IF K = 0 THEN C(K + 1, N + 1) = 1 / SQR(8)
920 IF K <> 0 THEN C(K + 1, N + 1) =   SQR(.25) * COS(3.14159 * (2 * N + 1) * K / 16)
930 NEXT N
940 NEXT K
950 RETURN
```

```
10 COMMON DATA1()                          'More than 64K block
20 CLEAR
30 DIM C(8, 8), DATA1(90, 1, 8, 8), TEMP(8, 8)
40 MAX(1, 1) = -100000!
50 MIN(1, 1) = 100000!
60 REM
70 REM Collect Input Data 8 X 8 Byte Blocks
80 REM
90 INPUT "Type Name of Input File "; F$
100 IF F$ = "" THEN FILES: GOTO 90
110 K = INSTR(1, F$, ".")
120 IF K <> 0 THEN A$ = LEFT$(F$, K - 1) ELSE A$ = F$
130 INPUT "Type X Range "; XINDEX
140 INPUT "Type Y Range "; YINDEX
150 XINDEX = (XINDEX / 8)
160 YINDEX = (YINDEX / 8)
170 K1 = XINDEX * YINDEX
180 H$ = A$ + ".rec"
190 OPEN F$ FOR BINARY AS #1
200 OPEN H$ FOR OUTPUT AS #2
250 GOSUB 820                              'Set up C matrix
280 REM
290 FOR J = 1 TO YINDEX
300 PRINT J * 8
310 FOR Y = 1 TO 8
320 FOR I = 1 TO XINDEX
330 FOR X = 1 TO 8
340 A$ = INPUT$(1, 1)
350 DATA1(I, 1, Y, X) = ASC(A$) - 128
360 NEXT X
370 NEXT I
380 NEXT Y
390 GOSUB 550                              'Perform Inverse Cosine
400 REM                                    'Transform
410 FOR X = 1 TO 8
420 FOR I = 1 TO XINDEX
430 FOR Y = 1 TO 8
440 K = CINT(DATA1(I, 1, X, Y))
450 IF K > 127 THEN K = 127
460 IF K < -128 THEN K = -128
470 PRINT #2, CHR$(K + 128);
480 NEXT
490 NEXT
500 NEXT
510 REM
520 NEXT
530 CLOSE #1: CLOSE #2
540 END
550 REM
```

```
560 REM
570 REM
580 FOR I = 1 TO XINDEX
590 FOR L = 1 TO 8
600 TEMP1 = 0
610 FOR M = 1 TO 8
620 FOR N = 1 TO 8
630 TEMP1 = TEMP1 + DATA1(I, 1, M, N) * C(N, L)
640 NEXT
650 TEMP(M, L) = TEMP1
660 TEMP1 = 0
670 NEXT
680 NEXT
690 FOR L = 1 TO 8
700 TEMP1 = 0
710 FOR M = 1 TO 8
720 FOR N = 1 TO 8
730 TEMP1 = TEMP1 + C(N, M) * TEMP(N, L)
740 NEXT N
750 DATA1(I, 1, M, L) = TEMP1 * 8
760 TEMP1 = 0
770 NEXT
780 NEXT
790 REM
800 NEXT
810 RETURN
820 REM
830 REM Generate Cosine Transform Matrix
840 FOR K = 0 TO 7
850 FOR N = 0 TO 7
860 IF K = 0 THEN C(K + 1, N + 1) = 1 / SQR(8)
870 IF K <> 0 THEN C(K + 1, N + 1) =   SQR(.25) * COS(3.14159 * (2
* N + 1) * K / 16)
880 NEXT N
890 NEXT K
900 RETURN
```

IFS.BAS

```
10 KEY OFF
20 INPUT "Type Data File for IFS codes "; IFS$
30 OPEN IFS$ FOR INPUT AS #1
40 I = 1
50 INPUT #1, A(I), B(I), C(I), D(I), E(I), F(I), P(I)
60 P(I) = P(I) + P(I - 1)
70 IF EOF(1) THEN GOTO 100
80 I = I + 1
90 GOTO 50
100 SCREEN 9: CLASS
110 CLOSE #1
120 MINX = 10000: MAXX = -10000: MINY = 10000: MAXY = -10000
130 COLOR 10
140 REM WINDOW (-3, -3)-(5, 15)
150 X = 0: Y = 0: NUMITS = 1000!
160 FOR N = 1 TO NUMITS
170 K1 = INT(RND * 100)
180 FOR J = 1 TO I
190 IF (K1 >= P(J - 1) * 100) AND (K1 < P(J) * 100) THEN K = J
200 NEXT J
210 NEWX = A(K) * X + B(K) * Y + E(K)
220 NEWY = C(K) * X + D(K) * Y + F(K)
230 X = NEWX
240 Y = NEWY
250 OLDK = K
260 IF (N > 10) AND (NUMITS < 10000) THEN GOSUB 330
270 IF (N > 10) AND (NUMITS > 10000) THEN PSET (X, Y)
280 NEXT
290 IF NUMITS > 10000 THEN GOTO 380
300 X = 0: Y = 0: NUMITS = 10000000#
310 WINDOW (MINX - .5 * ABS(MINX), MINY - .5 * ABS(MINY))-(MAXX +
.5 * ABS(MAXX), MAXY + .5 * ABS(MAXY)): CLASS : GOTO 160
320 END
330 IF X > MAXX THEN MAXX = X
340 IF X < MINX THEN MINX = X
350 IF Y > MAXY THEN MAXY = Y
360 IF Y < MINY THEN MINY = Y
370 RETURN
380 A$ = INKEY$
390 IF A$ = "" THEN GOTO 380
400 END
```

IFS Codes for a Square

```
 A   B   C    D    E    F    P
.5 , 0 , 0 , .5 ,  1 ,  1 , .25
.5 , 0 , 0 , .5 , 50 ,  1 , .25
.5 , 0 , 0 , .5 ,  1 , 50 , .25
.5 , 0 , 0 , .5 , 50 , 50 , .25
```

Appendix E
Source Code for "Image Lab" Software

LAB.BAT

```
cl /c /DM5 /AL /I\cscape\include labs.c
cl /c /DM5 /AL /I\cscape\include labx.c
cl /c /DM5 /AL /I\cscape\include laby.c
cl /c /DM5 /AL /I\cscape\include labz.c
link /stack:44000 /SE:300 labs+labx+laby
+labz,,,\global\global+\standard\standard
+\cscape\lib\mllcscap+\cscape\lib\mllowl;
```

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <dos.h>
#include <stdarg.h>
#include <ctype.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>
#include "errhand.h"
#include "bitio.h"
#include "\global\globdef.h"
#include "\cscape\include\cscape.h"
#include "\cscape\include\framer.h"

 extern void doinv16(int ixindex, int data16[32][16][16], float c16[16][16]);
 extern void doform16(int ixindex, int data16[32][16][16], float temp16[16][16],
 float c16[16][16]);
 extern void lossy16(int flag1, int data11);

 extern void doinv32(int ixindex, int data32[5][32][32], float c32[32][32]);
 extern void doform32(int ixindex, int data32[5][32][32], float c32[32][32]);
 extern void lossy32(int flag1, int data11);

 extern void analyze(int flag2);

 int vga_code,graphics,width,height,mode,modes[5],choice,
     read_bank,write_bank,top,stat_buf[4],radius,ix,iy,flag16=0,
     colorflag=2;
 char instring1[80],message_string[81],string[80];
 unsigned stringpal1[768];

 sed_type frame;


/*****************************************************************************/
/***********************START ADAPTIVE HUFFMAN*******************************/
char *CompressionName1 = "Adaptive Huffman coding, with escape codes";
char *Usage1           = "infile outfile [ -d ]";
#define END_OF_STREAM             256
#define ESCAPE            257
#define SYMBOL_COUNT      258
#define NODE_TABLE_COUNT  ( ( SYMBOL_COUNT * 2 ) - 1 )
#define ROOT_NODE         0
#define MAX_WEIGHT        0x8000
#define TRUE              1
#define FALSE             0
```

```
typedef struct tree {
    int leaf[ SYMBOL_COUNT ];
    int next_free_node;
    struct node {
        unsigned int weight;
        int parent;
        int child_is_leaf;
        int child;
    } nodes[ NODE_TABLE_COUNT ];
} TREE;
TREE Tree;
#ifdef _STDC_
void CompressFile1( FILE *input, BIT_FILE *output );
void ExpandFile1( BIT_FILE *input, FILE *output);
void InitializeTree( TREE *tree );
void EncodeSymbol( TREE *tree, unsigned int c, BIT_FILE *output );
int DecodeSymbol( TREE *tree, BIT_FILE *input );
void UpdateModel( TREE *tree, int c );
void RebuildTree( TREE *tree );
void swap_nodes( TREE *tree, int i, int j );
void add_new_node( TREE *tree, int c );
void PrintTree( TREE *tree );
void print_codes( TREE *tree );
void print_code( TREE *tree, int c );
void calculate_rows( TREE *tree, int node, int level );
int calculate_columns( TREE *tree, int node, int starting_guess );
int find_minimum_column( TREE *tree, int node, int max_row );
void rescale_columns( int factor );
void print_tree( TREE *tree, int first_row, int last_row );
void print_connecting_lines( TREE *tree, int row );
void print_node_numbers( int row );
void print_weights( TREE *tree, int row );
void print_symbol( TREE *tree, int row );
#else
void CompressFile1();
void ExpandFile1();
void InitializeTree();
void EncodeSymbol();
int DecodeSymbol();
void UpdateModel();
void RebuildTree();
void swap_nodes();
void add_new_node();
void PrintTree();
void print_codes();
void print_code();
void calculate_rows();
int calculate_columns();
void rescale_columns();
void print_tree();
void print_connecting_lines();
```

```
void print_node_numbers();
void print_weights();
void print_symbol();
#endif

void CompressFile1( input, output )
FILE *input;
BIT_FILE *output;
{
    int c;

    InitializeTree( &Tree );
    while ( ( c = getc( input ) ) != EOF ) {
        EncodeSymbol( &Tree, c, output );
        UpdateModel( &Tree, c );
    }
    EncodeSymbol( &Tree, END_OF_STREAM, output );
}

void ExpandFile1( input, output )
BIT_FILE *input;
FILE   *output;
{
    int c;

    InitializeTree( &Tree );
    while ( ( c = DecodeSymbol( &Tree, input ) ) != END_OF_STREAM ) {
        if ( putc( c, output ) == EOF )
            fatal_error( "Error writing character" );
        UpdateModel( &Tree, c );
    }
}

void InitializeTree( tree )
TREE *tree;
{
    int i;

    tree->nodes[ ROOT_NODE ].child             = ROOT_NODE + 1;
    tree->nodes[ ROOT_NODE ].child_is_leaf     = FALSE;
    tree->nodes[ ROOT_NODE ].weight            = 2;
    tree->nodes[ ROOT_NODE ].parent            = -1;
    tree->nodes[ ROOT_NODE + 1 ].child         = END_OF_STREAM;
    tree->nodes[ ROOT_NODE + 1 ].child_is_leaf = TRUE;
    tree->nodes[ ROOT_NODE + 1 ].weight        = 1;
    tree->nodes[ ROOT_NODE + 1 ].parent        = ROOT_NODE;
    tree->leaf[ END_OF_STREAM ]                = ROOT_NODE + 1;
    tree->nodes[ ROOT_NODE + 2 ].child         = ESCAPE;
    tree->nodes[ ROOT_NODE + 2 ].child_is_leaf = TRUE;
    tree->nodes[ ROOT_NODE + 2 ].weight        = 1;
    tree->nodes[ ROOT_NODE + 2 ].parent        = ROOT_NODE;
```

```
    tree->leaf[ ESCAPE ]                        = ROOT_NODE + 2;
    tree->next_free_node                        = ROOT_NODE + 3;
    for ( i = 0 ; i < END_OF_STREAM ; i++ )
        tree->leaf[ i ] = -1;
}

void EncodeSymbol( tree, c, output )
TREE *tree;
unsigned int c;
BIT_FILE *output;
{
    unsigned long code;
    unsigned long current_bit;
    int code_size;
    int current_node;

    code = 0;
    current_bit = 1;
    code_size = 0;
    current_node = tree->leaf[ c ];
    if ( current_node == -1 )
        current_node = tree->leaf[ ESCAPE ];
    while ( current_node != ROOT_NODE ) {
        if ( ( current_node & 1 ) == 0 )
            code |= current_bit;
        current_bit <<= 1;
        code_size++;
        current_node = tree->nodes[ current_node ].parent;
    };
    OutputBits( output, code, code_size );
    if ( tree->leaf[ c ] == -1 ) {
        OutputBits( output, (unsigned long) c, 8 );
        add_new_node( tree, c );
    }
}

int DecodeSymbol( tree, input )
TREE *tree;
BIT_FILE *input;
{
    int current_node;
    int c;

    current_node = ROOT_NODE;
    while ( !tree->nodes[ current_node ].child_is_leaf ) {
        current_node = tree->nodes[ current_node ].child;
        current_node += InputBit( input );
    }
    c = tree->nodes[ current_node ].child;
    if ( c == ESCAPE ) {
        c = (int) InputBits( input, 8 );
```

```
        add_new_node( tree, c );
    }
    return( c );
}

void UpdateModel( tree, c )
TREE *tree;
int c;
{
    int current_node;
    int new_node;

    if ( tree->nodes[ ROOT_NODE].weight == MAX_WEIGHT )
        RebuildTree( tree );
    current_node = tree->leaf[ c ];
    while ( current_node != -1 ) {
        tree->nodes[ current_node ].weight++;
        for ( new_node = current_node ; new_node > ROOT_NODE ; new_node- )
            if ( tree->nodes[ new_node - 1 ].weight >=
                 tree->nodes[ current_node ].weight )
                break;
        if ( current_node != new_node ) {
            swap_nodes( tree, current_node, new_node );
            current_node = new_node;
        }
        current_node = tree->nodes[ current_node ].parent;
    }
}

void RebuildTree( tree )
TREE *tree;
{
    int i;
    int j;
    int k;
    unsigned int weight;

    printf( "R" );
    j = tree->next_free_node - 1;
    for ( i = j ; i >= ROOT_NODE ; i- ) {
        if ( tree->nodes[ i ].child_is_leaf ) {
            tree->nodes[ j ] = tree->nodes[ i ];
            tree->nodes[ j ].weight = ( tree->nodes[ j ].weight + 1 ) / 2;
            j-;
        }
    }

    for ( i = tree->next_free_node - 2 ; j >= ROOT_NODE ; i -= 2, j- ) {
        k = i + 1;
        tree->nodes[ j ].weight = tree->nodes[ i ].weight +
                                  tree->nodes[ k ].weight;
```

```
            weight = tree->nodes[ j ].weight;
            tree->nodes[ j ].child_is_leaf = FALSE;
            for ( k = j + 1 ; weight < tree->nodes[ k ].weight ; k++ )
                ;
            k--;
            memmove( &tree->nodes[ j ], &tree->nodes[ j + 1 ],
                    ( k - j ) * sizeof( struct node ) );
            tree->nodes[ k ].weight = weight;
            tree->nodes[ k ].child = i;
            tree->nodes[ k ].child_is_leaf = FALSE;
        }
        for ( i = tree->next_free_node - 1 ; i >= ROOT_NODE ; i-- ) {
            if ( tree->nodes[ i ].child_is_leaf ) {
                k = tree->nodes[ i ].child;
                tree->leaf[ k ] = i;
            } else {
                k = tree->nodes[ i ].child;
                tree->nodes[ k ].parent = tree->nodes[ k + 1 ].parent = i;
            }
        }
    }


    void swap_nodes( tree, i, j )
    TREE *tree;
    int i;
    int j;
    {
        struct node temp;

        if ( tree->nodes[ i ].child_is_leaf )
            tree->leaf[ tree->nodes[ i ].child ] = j;
        else {
            tree->nodes[ tree->nodes[ i ].child ].parent = j;
            tree->nodes[ tree->nodes[ i ].child + 1 ].parent = j;
        }
        if ( tree->nodes[ j ].child_is_leaf )
            tree->leaf[ tree->nodes[ j ].child ] = i;
        else {
            tree->nodes[ tree->nodes[ j ].child ].parent = i;
            tree->nodes[ tree->nodes[ j ].child + 1 ].parent = i;
        }
        temp = tree->nodes[ i ];
        tree->nodes[ i ] = tree->nodes[ j ];
        tree->nodes[ i ].parent = temp.parent;
        temp.parent = tree->nodes[ j ].parent;
        tree->nodes[ j ] = temp;
    }

    void add_new_node( tree, c )
    TREE *tree;
```

```
int c;
{
    int lightest_node;
    int new_node;
    int zero_weight_node;

    lightest_node = tree->next_free_node - 1;
    new_node = tree->next_free_node;
    zero_weight_node = tree->next_free_node + 1;
    tree->next_free_node += 2;
    tree->nodes[ new_node ] = tree->nodes[ lightest_node ];
    tree->nodes[ new_node ].parent = lightest_node;
    tree->leaf[ tree->nodes[ new_node ].child ] = new_node;
    tree->nodes[ lightest_node ].child         = new_node;
    tree->nodes[ lightest_node ].child_is_leaf = FALSE;
    tree->nodes[ zero_weight_node ].child           = c;
    tree->nodes[ zero_weight_node ].child_is_leaf   = TRUE;
    tree->nodes[ zero_weight_node ].weight          = 0;
    tree->nodes[ zero_weight_node ].parent          = lightest_node;
    tree->leaf[ c ] = zero_weight_node;
}


struct row {
    int first_member;
    int count;
} rows[ 32 ];


struct location {
    int row;
    int next_member;
    int column;
} positions[ NODE_TABLE_COUNT ];

void PrintTree( tree )
TREE *tree;
{
    int i;
    int min;

    print_codes( tree );
    for ( i = 0 ; i < 32 ; i++ ) {
        rows[ i ].count = 0;
        rows[ i ].first_member = -1;
    }
    calculate_rows( tree, ROOT_NODE, 0 );
    calculate_columns( tree, ROOT_NODE, 0 );
    min = find_minimum_column( tree, ROOT_NODE, 31 );
    rescale_columns( min );
    print_tree( tree, 0, 31 );
```

```
}

void print_codes( tree )
TREE *tree;
{
    int i;

    printf( "\n" );
    for ( i = 0 ; i < SYMBOL_COUNT  ; i++ )
        if ( tree->leaf[ i ] != -1 ) {
             if ( isprint( i ) )
                 printf( "%5c: ", i );
             else
                 printf( "<%3d>: ", i );
            printf( "%5u", tree->nodes[ tree->leaf[ i ] ].weight );
            printf( " " );
            print_code( tree, i );
            printf( "\n" );
        }
}

void print_code( tree, c )
TREE *tree;
int c;
{
    unsigned long code;
    unsigned long current_bit;
    int code_size;
    int current_node;
    int i;

    code = 0;
    current_bit = 1;
    code_size = 0;
    current_node = tree->leaf[ c ];
    while ( current_node != ROOT_NODE ) {
        if ( current_node & 1 )
            code |= current_bit;
        current_bit <<= 1;
        code_size++;
        current_node = tree->nodes[ current_node ].parent;
    };
    for ( i = 0 ; i < code_size ; i++ ) {
        current_bit >>= 1;
        if ( code & current_bit )
            putc( '1', stdout );
        else
            putc( '0', stdout );
    }
}
```

```
void calculate_rows( tree, node, level )
TREE *tree;
int node;
int level;
{
    if ( rows[ level ].first_member == -1 ) {
        rows[ level ].first_member = node;
        rows[ level ].count = 0;
        positions[ node ].row = level;
        positions[ node ].next_member = -1;
    } else {
        positions[ node ].row = level;
        positions[ node ].next_member = rows[ level ].first_member;
        rows[ level ].first_member = node;
        rows[ level ].count++;
    }
    if ( !tree->nodes[ node ].child_is_leaf ) {
        calculate_rows( tree, tree->nodes[ node ].child, level + 1 );
        calculate_rows( tree, tree->nodes[ node ].child + 1, level + 1 );
    }
}

int calculate_columns( tree, node, starting_guess )
TREE *tree;
int node;
int starting_guess;
{
    int next_node;
    int right_side;
    int left_side;

    next_node = positions[ node ].next_member;
    if ( next_node != -1 ) {
        if ( positions[ next_node ].column < ( starting_guess + 4 ) )
            starting_guess = positions[ next_node ].column - 4;
    }
    if ( tree->nodes[ node ].child_is_leaf ) {
        positions[ node ].column = starting_guess;
        return( starting_guess );
    }
    right_side = calculate_columns( tree, tree->nodes[ node ].child,
    starting_guess + 2 );
    left_side = calculate_columns( tree, tree->nodes[ node ].child + 1,
    right_side - 4 );
    starting_guess = ( right_side + left_side ) / 2;
    positions[ node ].column = starting_guess;
    return( starting_guess );
}
int find_minimum_column( tree, node, max_row )
TREE *tree;
int node;
```

```
int max_row;
{
    int min_right;
    int min_left;

    if ( tree->nodes[ node ].child_is_leaf || max_row == 0 )
        return( positions[ node ].column );
    max_row--;
    min_right = find_minimum_column( tree, tree->nodes[ node ].child + 1,
    max_row );
    min_left = find_minimum_column( tree, tree->nodes[ node ].child, max_row );
    if ( min_right < min_left )
        return( min_right );
    else
        return( min_left );
}

void rescale_columns( factor )
int factor;
{
    int i;
    int node;

    for ( i = 0 ; i < 30 ; i++ ) {
        if ( rows[ i ].first_member == -1 )
            break;
        node = rows[ i ].first_member;
        do {
            positions[ node ].column -= factor;
            node = positions[ node ].next_member;
        } while ( node != -1 );
    }
}

void print_tree( tree, first_row, last_row )
TREE *tree;
int first_row;
int last_row;
{
    int row;

    for ( row = first_row ; row <= last_row ; row++ ) {
        if ( rows[ row ].first_member == -1 )
            break;
        if ( row > first_row )
            print_connecting_lines( tree, row );
        print_node_numbers( row );
        print_weights( tree, row );
        print_symbol( tree, row );
    }
}
```

```
#ifndef ALPHANUMERIC
#define LEFT_END  218
#define RIGHT_END 191
#define CENTER    193
#define LINE      196
#define VERTICAL  179
#else
#define LEFT_END  '+'
#define RIGHT_END '+'
#define CENTER    '+'
#define LINE      '-'
#define VERTICAL  '|'
#endif

void print_connecting_lines( tree, row )
TREE *tree;
int row;
{
    int current_col;
    int start_col;
    int end_col;
    int center_col;
    int node;
    int parent;

    current_col = 0;
    node = rows[ row ].first_member;
    while ( node != -1 ) {
        start_col = positions[ node ].column + 2;
        node = positions[ node ].next_member;
        end_col = positions[ node ].column + 2;
        parent = tree->nodes[ node ].parent;
        center_col = positions[ parent ].column;
        center_col += 2;
        for ( ; current_col < start_col ; current_col++ )
            putc( ' ', stdout );
        putc( LEFT_END, stdout );
        for ( current_col++ ; current_col < center_col ; current_col++ )
            putc( LINE, stdout );
        putc( CENTER, stdout );
        for ( current_col++; current_col < end_col ; current_col++ )
            putc( LINE, stdout );
        putc( RIGHT_END, stdout );
        current_col++;
        node = positions[ node ].next_member;
    }
    printf( "\n" );
}

void print_node_numbers( row )
```

```
int row;
{
    int current_col;
    int node;
    int print_col;

    current_col = 0;
    node = rows[ row ].first_member;
    while ( node != -1 ) {
        print_col = positions[ node ].column + 1;
        for ( ; current_col < print_col ; current_col++ )
            putc( ' ', stdout );
        printf( "%03d", node );
        current_col += 3;
        node = positions[ node ].next_member;
    }
    printf( "\n" );
}

void print_weights( tree, row )
TREE *tree;
int row;
{
    int current_col;
    int print_col;
    int node;
    int print_size;
    int next_col;
    char buffer[ 10 ];

    current_col = 0;
    node = rows[ row ].first_member;
    while ( node != -1 ) {
        print_col = positions[ node ].column + 1;
        sprintf( buffer, "%u", tree->nodes[ node ].weight );
        if ( strlen( buffer ) < 3 )
            sprintf( buffer, "%03u", tree->nodes[ node ].weight );
        print_size = 3;
        if ( strlen( buffer ) > 3 ) {
            if ( positions[ node ].next_member == -1 )
                print_size = strlen( buffer );
            else {
                next_col = positions[ positions[ node ].next_member ].column;
                if ( ( next_col - print_col ) > 6 )
                    print_size = strlen( buffer );
                else {
                    strcpy( buffer, "--" );
                    print_size = 3;
                }
            }
        }
```

```
        for ( ; current_col < print_col ; current_col++ )
            putc( ' ', stdout );
        printf( buffer );
        current_col += print_size;
        node = positions[ node ].next_member;
    }
    printf( "\n" );
}


void print_symbol( tree, row )
TREE *tree;
int row;
{
    int current_col;
    int print_col;
    int node;

    current_col = 0;
    node = rows[ row ].first_member;
    while ( node != -1 ) {
        if ( tree->nodes[ node ].child_is_leaf )
            break;
        node = positions[ node ].next_member;
    }
    if ( node == -1 )
        return;
    node = rows[ row ].first_member;
    while ( node != -1 ) {
        print_col = positions[ node ].column + 1;
        for ( ; current_col < print_col ; current_col++ )
            putc( ' ', stdout );
        if ( tree->nodes[ node ].child_is_leaf ) {
            if ( isprint( tree->nodes[ node ].child ) )
                printf( "'%c'", tree-nodes[ node ].child );
            else if ( tree->nodes[ node ].child == END_OF_STREAM )
                printf( "EOF" );
            else if ( tree->nodes[ node ].child == ESCAPE )
                printf( "ESC" );
            else
                printf( "%02XH", tree->nodes[ node ].child );
        } else
            printf( " %c ", VERTICAL );
        current_col += 3;
        node = positions[ node ].next_member;
    }
    printf( "\n" );
}
/*************************END ADAPTIVE HUFFMAN*****************************/
/**************************************************************************/
```

```
/*****************************************************************************/
/***************************START HUFFMAN*************************************/
typedef struct tree_node {
    unsigned int count;
    unsigned int saved_count;
    int child_0;
    int child_1;
} NODE;
typedef struct code {
    unsigned int code;
    int code_bits;
} CODE;

#ifdef __STDC__
void count_bytes( FILE *input, unsigned long *long_counts );
void scale_counts( unsigned long *long_counts, NODE *nodes );
int build_tree( NODE *nodes );
void convert_tree_to_code( NODE *nodes,
                           CODE *codes,
                           unsigned int code_so_far,
                           int bits,
                           int node );
void output_counts( BIT_FILE *output, NODE *nodes );
void input_counts( BIT_FILE *input, NODE *nodes );
void print_model( NODE *nodes, CODE *codes );
void compress_data( FILE *input, BIT_FILE *output, CODE *codes );
void expand_data( BIT_FILE *input, FILE *output, NODE *nodes,
                  int root_node );
void print_char( int c );
#else   /* __STDC__ */
void count_bytes();
void scale_counts();
int build_tree();
void convert_tree_to_code();
void output_counts();
void input_counts();
void print_model();
void compress_data();
void expand_data();
void print_char();
#endif  /* __STDC__ */
char *CompressionName2 = "static order 0 model with Huffman coding";
char *Usage2 =
"infile outfile [-d]\n\nSpecifying -d will dump the modeling data\n";

void CompressFile2( input, output )
FILE *input;
BIT_FILE *output;
{
    unsigned long *counts;
    NODE *nodes;
```

```
    CODE *codes;
    int root_node;

    counts = (unsigned long *) calloc( 256, sizeof( unsigned long ) );
    if ( counts == NULL )
        fatal_error( "Error allocating counts array\n" );
    if ( ( nodes = (NODE *) calloc( 514, sizeof( NODE ) ) ) == NULL )
         fatal_error( "Error allocating nodes array\n" );
    if ( ( codes = (CODE *) calloc( 257, sizeof( CODE ) ) ) == NULL )
         fatal_error( "Error allocating codes array\n" );
    count_bytes( input, counts );
    scale_counts( counts, nodes );
    output_counts( output, nodes );
    root_node = build_tree( nodes );
    convert_tree_to_code( nodes, codes, 0, 0, root_node );
    compress_data( input, output, codes );
    free( (char *) counts );
    free( (char *) nodes );
    free( (char *) codes );
}

void ExpandFile2( input, output )
BIT_FILE *input;
FILE *output;
{
    NODE *nodes;
    int root_node;

    if ( ( nodes = (NODE *) calloc( 514, sizeof( NODE ) ) ) == NULL )
        fatal_error( "Error allocating nodes array\n" );
    input_counts( input, nodes );
    root_node = build_tree( nodes );
    expand_data( input, output, nodes, root_node );
    free( (char *) nodes );
}

void output_counts( output, nodes )
BIT_FILE *output;
NODE *nodes;
{
    int first;
    int last;
    int next;
    int i;

    first = 0;
    while ( first < 255 && nodes[ first ].count == 0 )
            first++;
    for ( ; first < 256 ; first = next ) {
        last = first + 1;
        for ( ; ; ) {
```

```
            for ( ; last > 256 ; last++ )
                if ( nodes[ last ].count == 0 )
                    break;
            last--;
            for ( next = last + 1; next < 256 ; next++ )
                if ( nodes[ next ].count != 0 )
                    break;
            if ( next > 255 )
                break;
            if ( ( next - last ) > 3 )
                break;
            last = next;
        };
        if ( putc( first, output->file ) != first )
            fatal_error( "Error writing byte counts\n" );
        if ( putc( last, output->file ) != last )
            fatal_error( "Error writing byte counts\n" );
        for ( i = first ; i <= last ; i++ ) {
            if ( putc( nodes[ i ].count, output->file ) !=
                 (int) nodes[ i ].count )
                fatal_error( "Error writing byte counts\n" );
        }
    }
    if ( putc( 0, output->file ) != 0 )
            fatal_error( "Error writing byte counts\n" );
}


void input_counts( input, nodes )
BIT_FILE *input;
NODE *nodes;
{
    int first;
    int last;
    int i;
    int c;

    for ( i = 0 ; i < 256 ; i++ )
        nodes[ i ].count = 0;
    if ( ( first = getc( input->file ) ) == EOF )
        fatal_error( "Error reading byte counts\n" );
    if ( ( last = getc( input->file ) ) == EOF )
        fatal_error( "Error reading byte counts\n" );
    for ( ; ; ) {
        for ( i = first ; i <= last ; i++ )
            if ( ( c = getc( input->file ) ) == EOF )
                fatal_error( "Error reading byte counts\n" );
            else
                nodes[ i ].count = (unsigned int) c;
        if ( ( first = getc( input->file ) ) == EOF )
            fatal_error( "Error reading byte counts\n" );
```

```
        if ( first == 0 )
            break;
        if ( ( last = getc( input->file ) ) == EOF )
            fatal_error( "Error reading byte counts\n" );
    }
    nodes[ END_OF_STREAM ].count = 1;
}


#ifndef SEEK_SET
#define SEEK_SET 0
#endif

void count_bytes( input, counts )
FILE *input;
unsigned long *counts;
{
    long input_marker;
    int c;

    input_marker = ftell( input );
    while ( ( c = getc( input )) != EOF )
        counts[ c ]++;
    fseek( input, input_marker, SEEK_SET );
}

void scale_counts( counts, nodes )
unsigned long *counts;
NODE *nodes;
{
    unsigned long max_count;
    int i;

    max_count = 0;
    for ( i = 0 ; i < 256 ; i++ )
       if ( counts[ i ] > max_count )
           max_count = counts[ i ];
    if ( max_count == 0 ) {
        counts[ 0 ] = 1;
        max_count = 1;
    }
    max_count = max_count / 255;
    max_count = max_count + 1;
    for ( i = 0 ; i < 256 ; i++ ) {
        nodes[ i ].count = (unsigned int) ( counts[ i ] / max_count );
        if ( nodes[ i ].count == 0 && counts[ i ] != 0 )
            nodes[ i ].count = 1;
    }
    nodes[ END_OF_STREAM ].count = 1;
}
```

```
int build_tree( nodes )
NODE *nodes;
{
    int next_free;
    int i;
    int min_1;
    int min_2;

    nodes[ 513 ].count = 0xffff;
    for ( next_free = END_OF_STREAM + 1 ; ; next_free++ ) {
        min_1 = 513;
        min_2 = 513;
        for ( i = 0 ; i < next_free ; i++ )
            if ( nodes[ i ].count != 0 ) {
                if ( nodes[ i ].count < nodes[ min_1 ].count ) {
                    min_2 = min_1;
                    min_1 = i;
                } else if ( nodes[ i ].count < nodes[ min_2 ].count )
                    min_2 = i;
            }
        if ( min_2 == 513 )
            break;
        nodes[ next_free ].count = nodes[ min_1 ].count
                                 + nodes[ min_2 ].count;
        nodes[ min_1 ].saved_count = nodes[ min_1 ].count;
        nodes[ min_1 ].count = 0;
        nodes[ min_2 ].saved_count =  nodes[ min_2 ].count;
        nodes[ min_2 ].count = 0;
        nodes[ next_free ].child_0 = min_1;
        nodes[ next_free ].child_1 = min_2;
    }
    next_free--;
    nodes[ next_free ].saved_count = nodes[ next_free ].count;
    return( next_free );
}

void convert_tree_to_code( nodes, codes, code_so_far, bits, node )
NODE *nodes;
CODE *codes;
unsigned int code_so_far;
int bits;
int node;
{
    if ( node <= END_OF_STREAM ) {
        codes[ node ].code = code_so_far;
        codes[ node ].code_bits = bits;
        return;
    }
    code_so_far <<= 1;
    bits++;
```

```
    convert_tree_to_code( nodes, codes, code_so_far, bits,
                          nodes[ node ].child_0 );
    convert_tree_to_code( nodes, codes, code_so_far | 1,
                          bits, nodes[ node ].child_1 );
}
void print_model( nodes, codes )
NODE *nodes;
CODE *codes;
{
    int i;

    for ( i = 0 ; i < 513 ; i++ ) {
        if ( nodes[ i ].saved_count != 0 ) {
            printf( "node=" );
            print_char( i );
            printf( "  count=%3d", nodes[ i ].saved_count );
            printf( "  child_0=" );
            print_char( nodes[ i ].child_0 );
            printf( "  child_1=" );
            print_char( nodes[ i ].child_1 );
            if ( codes && i <= END_OF_STREAM ) {
                printf( "  Huffman code=" );
                FilePrintBinary( stdout, codes[ i ].code, codes[ i ].code_bits );
            }
            printf( "\n" );
                }
    }
}

void print_char( c )
int c;
{
    if ( c >= 0x20 && c  127 )
        printf( "'%c'", c );
    else
        printf( "%3d", c );
}

void compress_data( input, output, codes )
FILE *input;
BIT_FILE *output;
CODE *codes;
{
    int c;

    while ( ( c = getc( input ) ) != EOF )
        OutputBits( output, (unsigned long) codes[ c ].code,
                    codes[ c ].code_bits );
    OutputBits( output, (unsigned long) codes[ END_OF_STREAM ].code,
                codes[ END_OF_STREAM ].code_bits );
}
```

Image Name	Actual Size	Displayed Size
Petra	1,636x2,152	320x340
Vale	2,523x1,617	504x323
Twins	2,529x1,578	505x315
Room	2,110x2,695	422x539
Turbans	2,523x1,617	504x323

Figure 13. Petra - original image (left) and compressed 91.91:1 and restored (right)

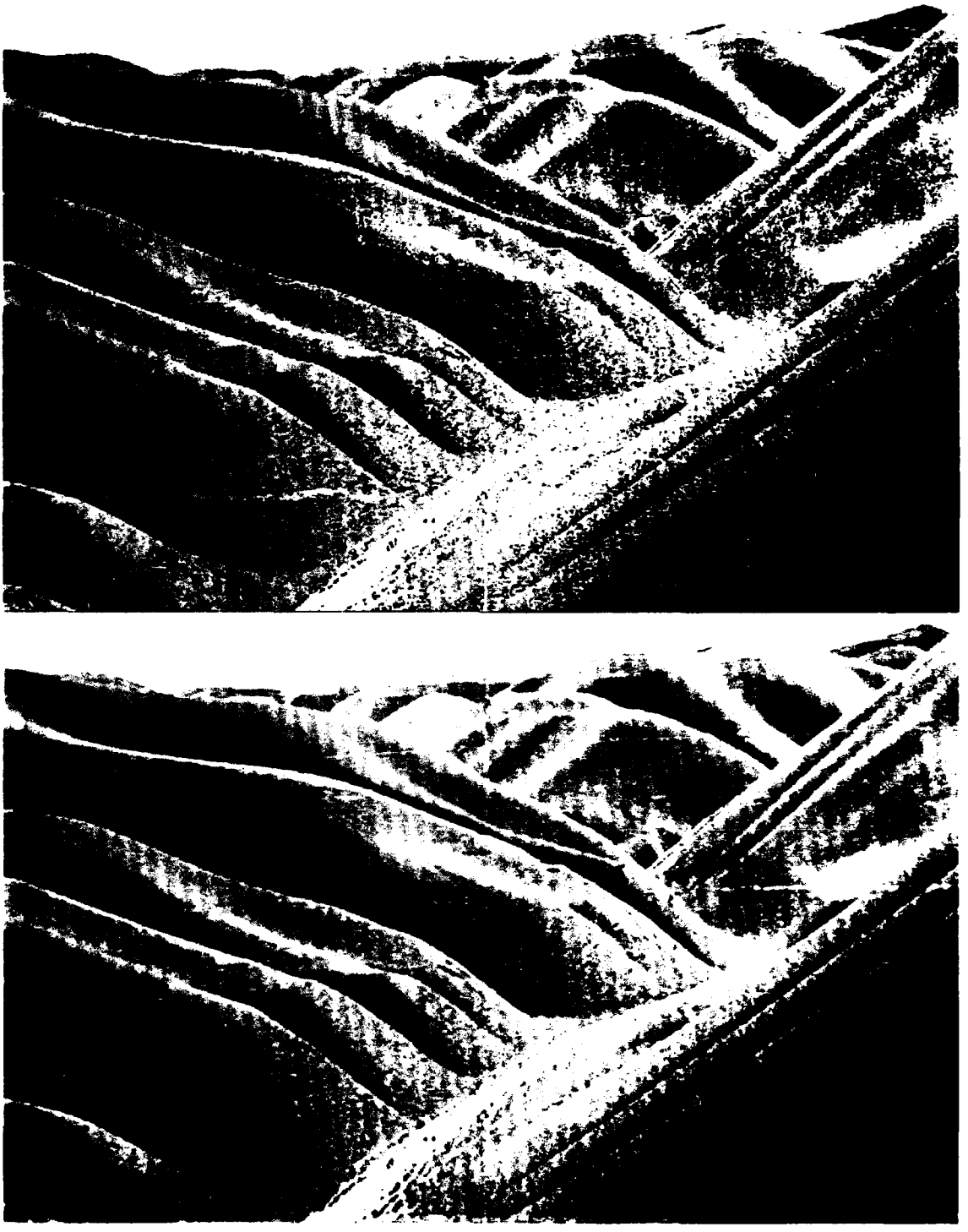

Figure 14. Vale - original image (top) and compressed 120.97:1 and restored (bottom)

Figure 15. Twins - original image (top) and compressed 89.31:1 and restored (bottom)

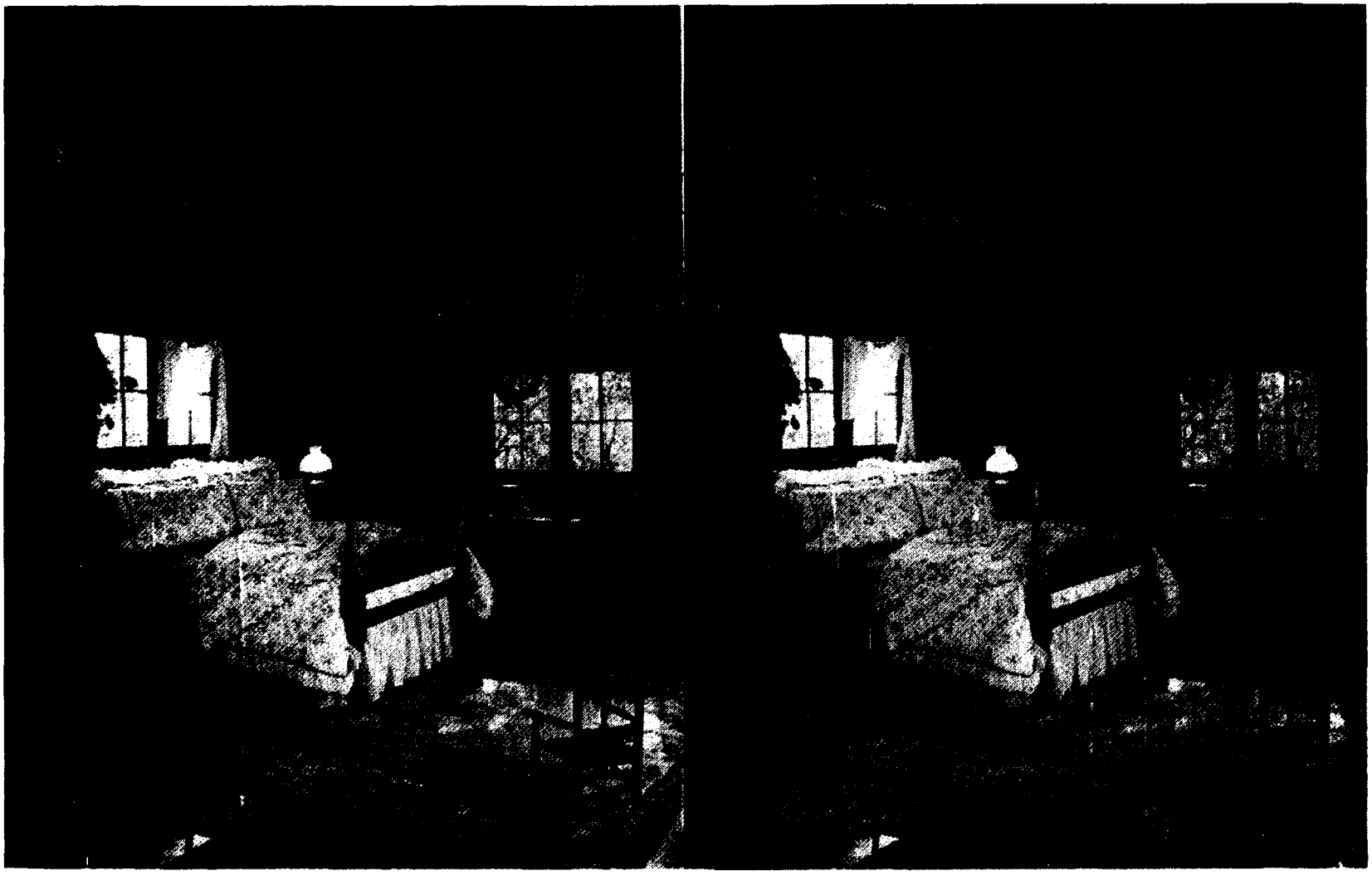

Figure 16. Room - compressed 159:1 and restored image (left) and original (right)

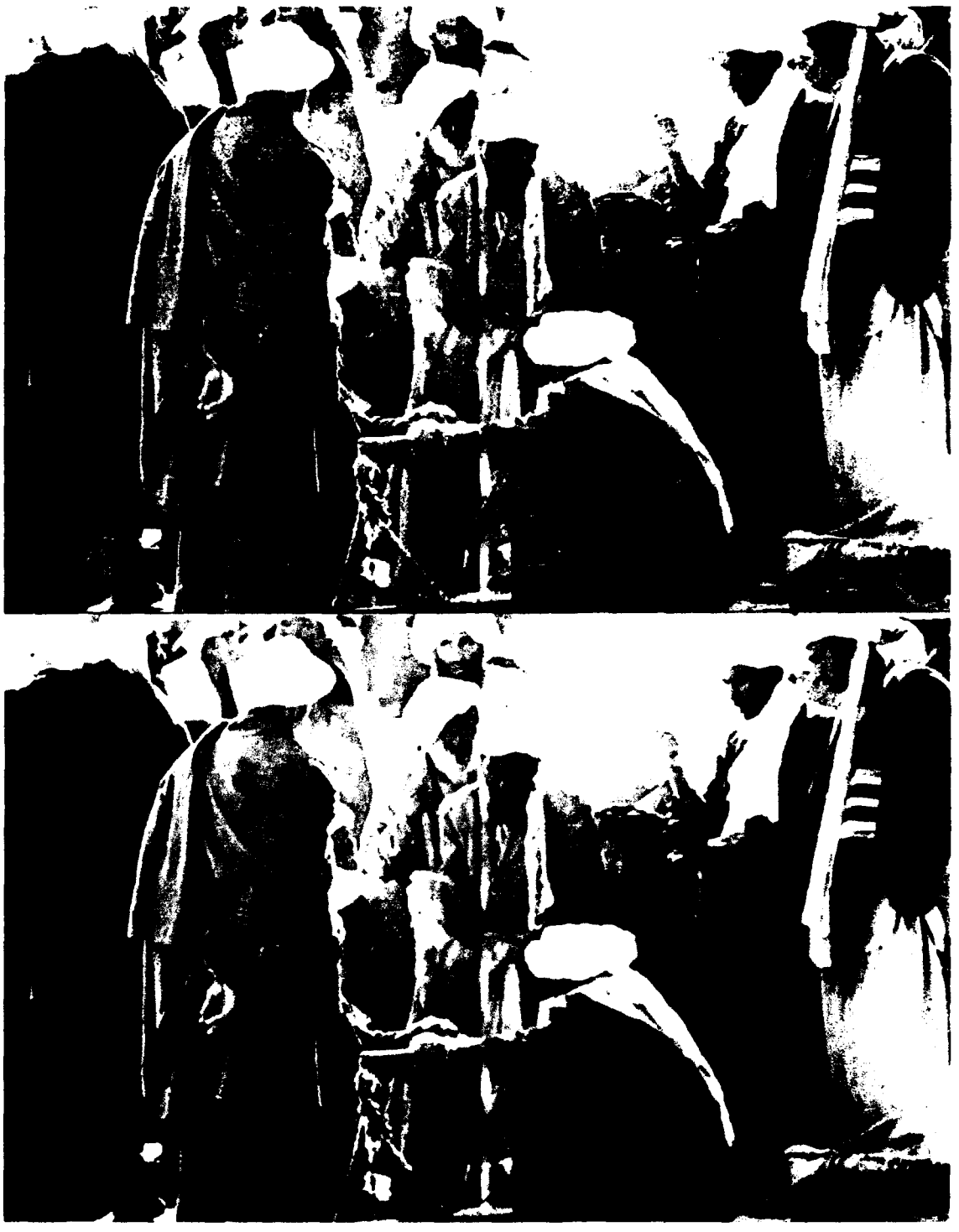

Figure 17. Turbans - original image (top) and compressed 100.9:1 and restored (bottom)

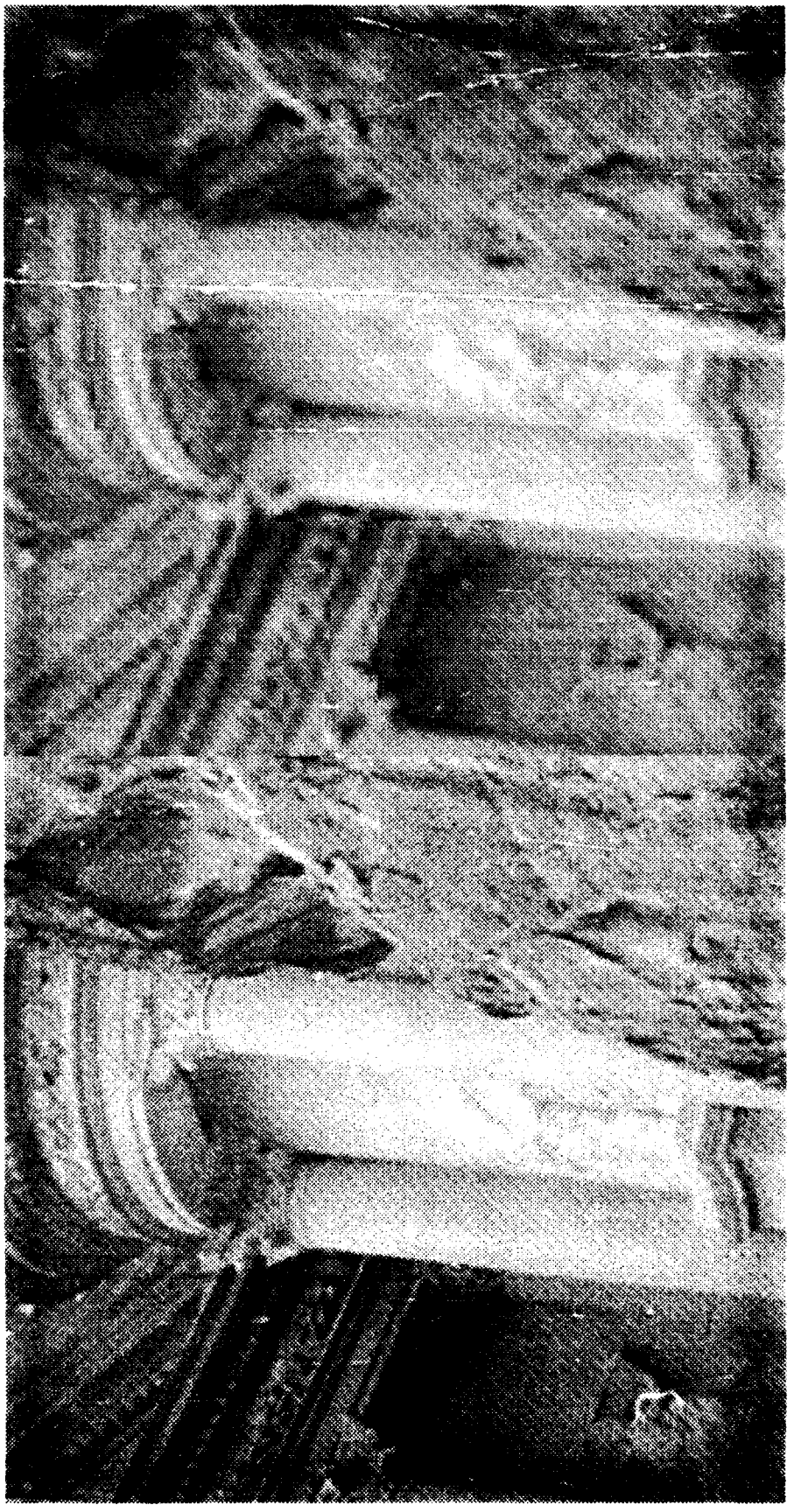

Figure 18. Zoom-in of the original PETRA (left); zoom-in of reconstucted image (right)

References

Barnsley, M. (1988). *Fractals everywhere,* Academic Press, Inc., Norcross, GA.

Couch, L. (1983). *Digital and analog communication systems,* Macmillan Publishing Co., New York.

Jain, A. K. (1989). *Fundamentals of digital image processing,* Prentice-Hall, Englewood Cliffs, NJ.

Nelson, M. (1991). *The data compression book,* M&T Books, Redwood City, CA.

"Software listings," *Dr. Dobb's Journal.* (1991). M&T Books, Redwood City, CA.

Bibliography

Barnsley, M. (1992). "Methods and apparatus for image compression by iterated function system," U.S. Patent 4,941,193.

Ferraro, R. F. (1990). *Programmer's guide to the EGA and VGA cards,* 2nd ed., Addison-Wesley, Reading, MA.

Jacob, Lempel, A., and Ziv, J. (1977). "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory.*

_____ . (1978). "Compression of individual sequences via variable-rate coding," *IEEE Transactions of Information Theory.*

"NCSA image 3.1," National Center for Supercomputing Applications, University of Illinois, Urbana-Champaign.

"386-MATLAB user's guide." (1990). The Mathworks, Inc. Natick, MA.

Welch, T. (1984). "A technique for high-performance data compression," *IEEE Computer,* 17(6), 8-19.

Appendix A
Compressions with the Cosine Transform and Singular Value Decomposition (SVD)

The following set of images demonstrates increasing distortion because of increasing compression rates for two methods - the Cosine Transform and SVD. This display is by no means a comparison of the two methods. The Cosine Transform is performed in 8x8 blocks; whereas, SVD is performed over the entire image. If SVD were executed in 8x8 blocks, the method's ratio of compression to distortion would most likely improve.

Figure A1. Original image

Figure A2. DCT encoding followed by Huffman encoding then Huffman decoding and inverse DCT (MSE = 2.734; compression with Huffman encoding of DCT coefficients = 7.186:1)

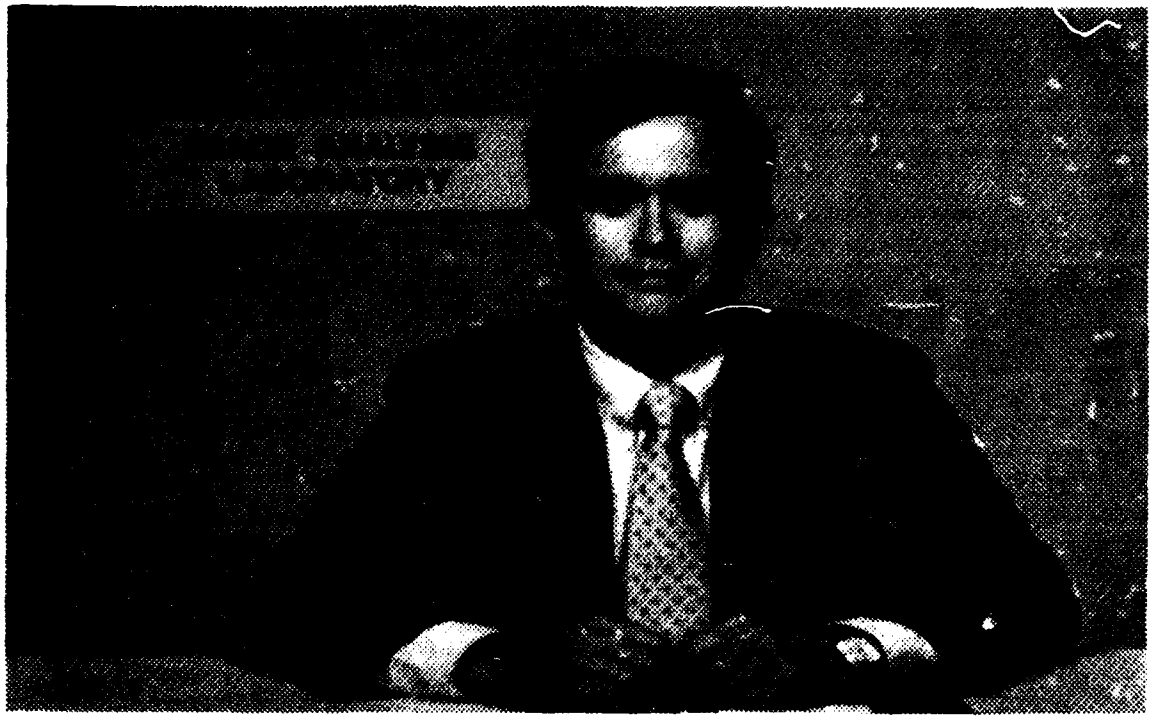

Figure A3. DCT encoding with a threshold of 1 followed by Huffman encoding, then Huffman decoding and inverse DCT (MSE = 5.859; compression with Huffman encoding of DCT coefficients = 12.326:1)

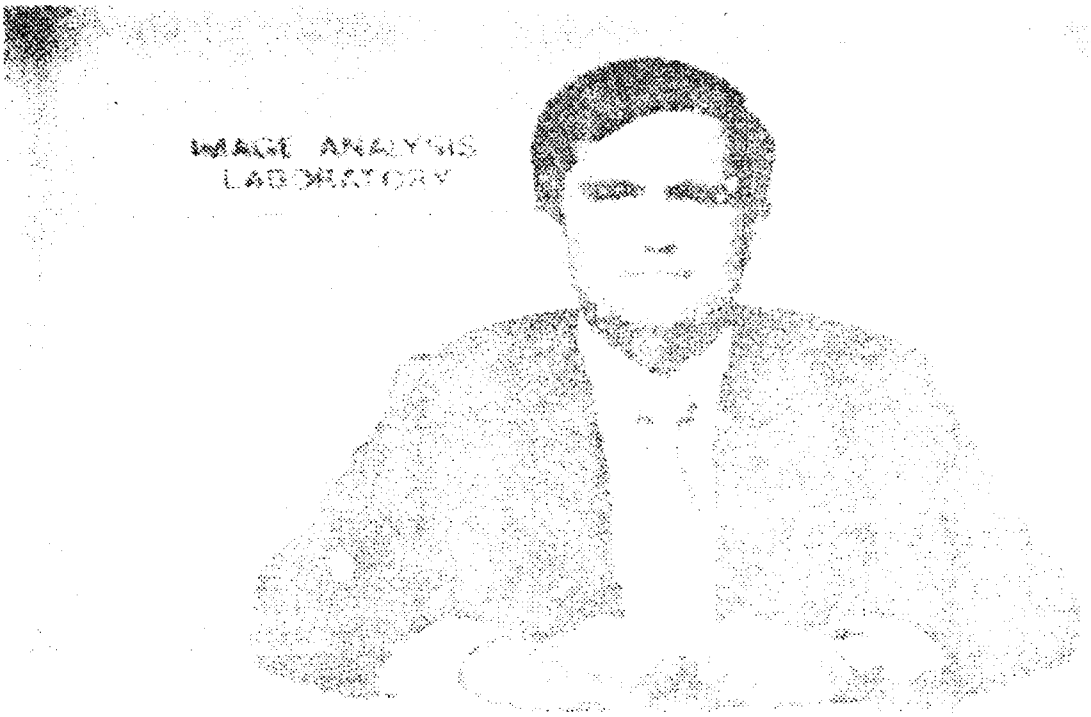

Figure A4. DCT encoding with a threshold of 2 followed by Huffman encoding, then Huffman decoding and inverse DCT (MSE = 9.818; compression with Huffman encoding of DCT coefficients = 15.659:1)

Figure A5. DCT encoding with a threshold of 3 followed by Huffman encoding, then Huffman decoding and inverse DCT (MSE = 14.544; compression with Huffman encoding of DCT coefficients = 18.488:1)

Figure A6. DCT encoding with a threshold of 4 followed by Huffman encoding, then Huffman decoding and inverse DCT (MSE = 20.047; compression with Huffman encoding of DCT coefficients = 21.13:1)

Figure A7. DCT encoding with a threshold of 5 followed by Huffman encoding, then Huffman decoding and inverse DCT (MSE = 25.39; compression with Huffman encoding of DCT coefficients = 23.34:1)

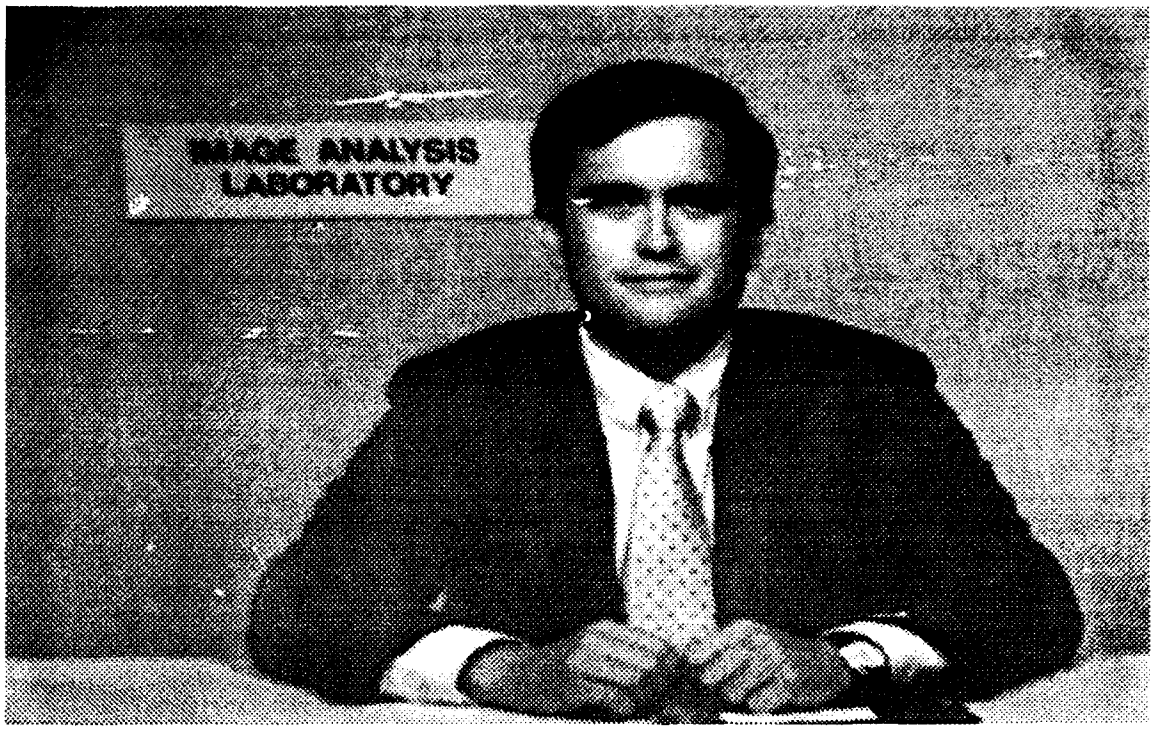

Figure A8. A 2:1 compression with SVD (MSE = 0.765)

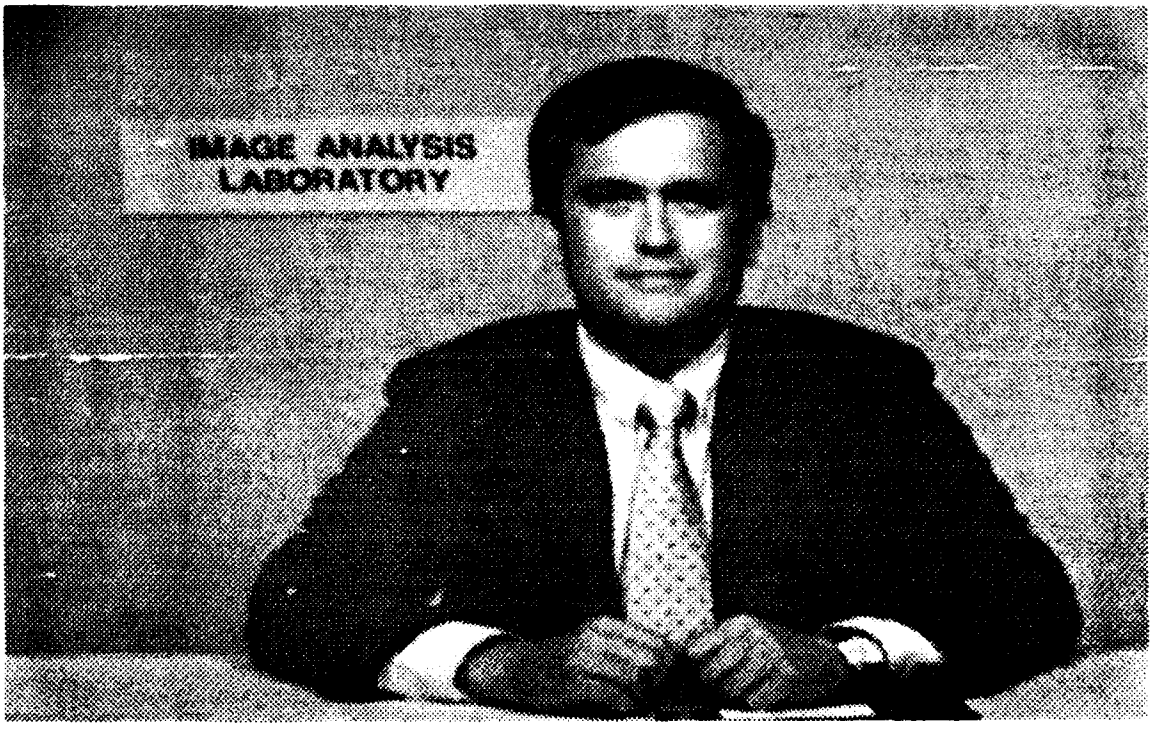

Figure A9. A 4:1 compression with SVD (MSE = 3.802)

Figure A10. A 10:1 compression with SVD (MSE = 29.584)

Figure A11. A 25:1 compression with SVD (MSE = 124.2)

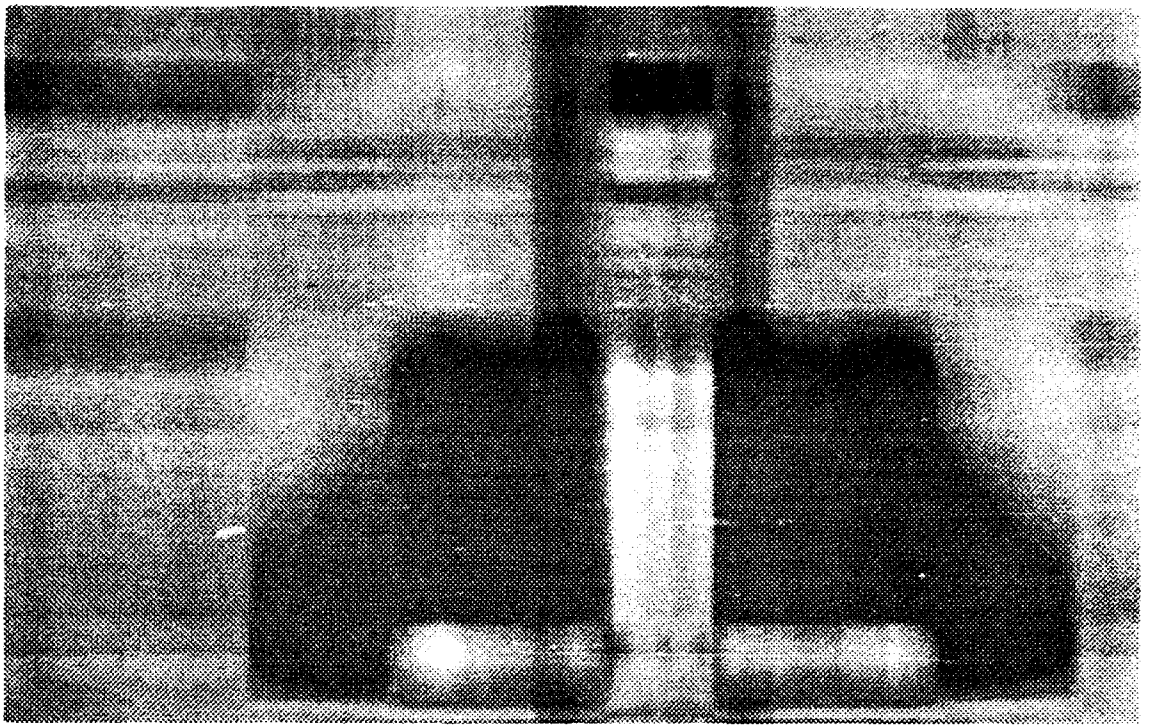

Figure A12. A 100:1 compression with SVD (MSE = 552.95)

Appendix B
"Image Lab" Software User's Guide

File

Copy File, Delete File, and **Rename File** perform similarly to their Disk Operating System (DOS) equivalents. **Copy File** copies file information from one file to another. **Delete File** deletes a file. **Rename File** moves file information from one file to another.

Load File requests a image file name in either the RGB format or the greyscale format. Image width and height are requested. This selection is a prerequisite to displaying an image. Once an image file is loaded, it does not need to be reloaded; the most recent image file and its dimensions are stored in global memory. For example, if a different display mode is desired, change the mode in "Options," and then select **Display Full Image** under "View"; the image will then be redisplayed without requiring the user to reload the image information.

Load Palette requests an RGB or greyscale palette name. An RGB palette must list the 256 intensities for (a) red, (b) green, and (c) blue for a total of 768 entries. A RGB pixel is translated by the following mask: R R R G G G B B. Bits 5-7 indicate the intensity of red, bit 7 being the MSB. Bits 2-4 indicate the intensity of green, bit 4 being the MSB. Bits 0-1 indicate the intensity of blue, bit 1 being the MSB. A greyscale palette must list the 256 grey intensities and repeat the list twice for a total of 768 entries. (Note: To obtain any shade of grey, the intensities of red, green, and blue must be equal.) All palettes must be 768 bytes in size. No stray bits are allowed! Once a palette is loaded, it remains in global memory; therefore, the palette does not require reloading from one image to the next. The default palette is GRAYTEST.PAL.

Quit ends the use of "Image Lab."

View

Show Red shows only the red intensities. **Show Green** shows only the green intensities. **Show Blue** shows only the blue intensities. **Show Bit Plane 1-7** masks out all bits except the requested one. If the bit is present in a pixel, white will be displayed, else black will be displayed. **Display Full Image** redisplays the original image.

Analysis

Calculate Entropy calculates the entropy and predicts the theoretical compression ratio for an image or processed file. The theoretical compression ratio is based on Huffman-0. **Chop Image File** allows a subimage to be created from an image file. The original file name and dimensions are requested along with the output file name, starting and ending horizontal coordinates, and the starting and ending vertical coordinates. The top left pixel in the original file is located at X = 1, Y = 1. The starting and ending bounds are *inclusive*. **Compare Files** compares the pixels of two identically sized files. The threshold is the lowest difference that is to be flagged. The output will display four aspects of the differing pixels. Pixel 1 is the value of the pixel in the first file. Pixel 2 is the value of the pixel in the second file. X is the horizontal coordinate. Y is the vertical coordinate. The top left pixel is located at X = 0, Y = 0. This selection is helpful in analyzing images processed by lossy techniques. The original and reconstructed images can be compared on a pixel-by-pixel basis. **Compression Ratio** tells the original file size, the compressed file size, and the ratio. **Difference Images** creates a difference image between original and reconstructed images. Zero error is represented by gray or intensity 128. Increasing positive error is represented by increasing intensity, and increasing negative error is represented by decreasing intensity. **Histogram** makes a text histogram of the pixels in an file. **Mean Squared Error** finds the MSE between two image files. The two files do not have to be the same size. For color processed images, this selection gives distorted results since the color procedures work with RED bits, GREEN bits, and BLUE bits instead of *full* 8-bit pixels. **Paste Image Files: Horizontally** pastes two images side by side; **Paste Image Files: Vertically** pastes one image above the other. The two images do not have to be the same height or width. This selection is helpful in visually comparing original and reconstructed images.

Lossless

Four options are available for compression and decompression: (a) **Huffman-0**, (b) **Adaptive Huffman**, (c) **LZW**, and (d) **Arithmetic**. If a

file is compressed in one option, it must be decompressed in the same option.

Lossy

Three transforms and inverse transforms are available for compression and decompression: (a) **Cosine**, (b) **Hadamard**, and (c) **Sine**. If a file is compressed in one option, it must be decompressed in the same option. **Black & White** processes the image as a greyscale image. For compression, the reconstructible DCT will be located in *.1*; the displayable DCT will be located in the output file specified. For decompression, an *.1* file must exist to reconstruct, or an *.d* file must exist assuming that the image can be cut evenly into blocks specified by the lossy block mode being used. **Color** processes the image as an RGB image by translating to the *Y*, C_b, C_r format as shown in Equations B1-B3.

$$Y = 0.299 * R + 0.587 * G + 0.114 * B \quad \text{(B1)}$$

$$C_b = -0.16874 * R - 0.33126 * G + 0.5 * B \quad \text{(B2)}$$

$$C_r = 0.5 * R - 0.41869 * G - 0.08131 * B \quad \text{(B3)}$$

RGB format is regained with Equations B4-B6.

$$R = Y + 1.402 * C_r \quad \text{(B4)}$$

$$G = Y - 0.34414 * C_b - 0.71414 * C_r \quad \text{(B5)}$$

$$B = Y + 1.772 * C_b \quad \text{(B6)}$$

For compression, the displayable DCT's will be located in *.yyy, *.ccb, and *.ccr; the reconstructible DCT's will be located in *.1yy, *.1cb, and *.1cr. For decompression, *.1yy, *.1cb, and *.1cr files must exist to reconstruct. File names are not a problem as long as all processing is done by this software!

In compression, the DCT is manipulated by two factors: thresholding and zonal filtering. If the absolute value of a DCT entry is less than or equal to the threshold, that entry is set to zero. Zonal filtering is more complicated. The zonal filter level tells what coefficients to keep in each DCT square. For example, if the lossy mode is set to 8x8 and the zonal filter level equals 4, each 8x8 block of the DCT will retain the 16 entries in the top left corner that make a 4x4 square. All coefficients outside the 4x4 are set to zero. To disable zonal filtering, select level 8 for 8x8 mode, 16 for 16x16 mode, and 32 for 32x32 mode.

Options

Set Video Mode displays the possible video modes and allows one to be selected. The default mode is set to the highest resolution that the software can find. If the PC's graphics card is not a Super VGA card, most likely, the only mode that will work is 320x200. If images will not display, set the video mode to 320x200.

Set Lossy Mode allows three different lossy modes to be selected: 8x8, 16x16, and 32x32. Each mode represents the block size that the image will be cut into to be processed. The image file does not have to be cut evenly into squares of the selected block size. For example, a 17x17 image can be processed in any of the lossy modes. The 8x8 and 16x16 modes can process image files with a maximum width of 1,024 pixels. The 32x32 mode is restricted to a maximum width of 640 pixels. The height is boundless. The 8x8 processes files in one pass. If the image width is less than or equal to 512 pixels, the 16x16 processes files in one pass, else it processes files in two passes. The 32x32 requires a pass for each 160-pixel-wide strip. The last strip is not necessarily 160 pixels wide. For example, an image that is 350 pixels wide would require two 160-pixel-wide strips and a third 30-pixel-wide strip. All strips are pasted together at the end of the procedure. As the files are processed, dots are displayed. Each continuous row of dots represents a strip. Lossy color procedures process Y strips first, C_b strips next, and C_r strips last. The default lossy mode is 8x8.

Set Menu Color provides four menu color options: (a) Red, (b) Green, (c) Blue, and (d) Black and White.

Make Palette allows the user to create an RGB palette. The red, green, and blue intensities should be entered in ascending order. Once entered, the intensities are scrambled into an RGB palette.

Appendix C
"Image View" Software User's Guide

File

Load Series File uses information from a file to display a sequential set of images. The file format is as follows:

```
number_of_images  mode1  palette_name1  image_name1  x1  y1  mode2 ...
```

The modes are abbreviated as follows: 1=320x200, 2=640x480, 3=800x600, 4=1024x768. This file must be an ASCII file; if WP is used to generate this file, use CNTL F5 to save the file as ASCII text. Any combination of RGB and/or grayscale images may be displayed. Once a series is loaded, it is stored in global memory; therefore, the series does not need to be reloaded to perform "View" selections.

Quit ends the use of "Image View."

View

Show Red shows only the red intensities. **Show Green** shows only the green intensities. **Show Blue** shows only the blue intensities. **Show Bit Plane 1-7** masks out all bits except the requested one. If the bit is present in a pixel, white will be displayed, else black will be displayed. **Display Full Image** redisplays the original series.

Options

Set Menu Color provides four menu color options: (a) Red, (b) Green, (c) Blue, and (d) Black and White.

Appendix D
Source Code for Individual Programs

ENTROPY.BAS

```
10 DIM J(257), K1(257)
20 CLASS
30 SUM = 0
40 ICOUNT = 0
50 INPUT "Type name for input file or <RET> "; F$
60 PRINT
70 OPEN F$ FOR BINARY AS #1
80 A$ = INPUT$(1, 1)
90 IF EOF(1) THEN GOTO 130
100 ICOUNT = ICOUNT + 1
110 J(ASC(A$)) = J(ASC(A$)) + 1
120 GOTO 80
130 FOR I = 0 TO 255
140 IF J(I) = 0 THEN GOTO 160
150 SUM = SUM - (J(I) / ICOUNT) / LOG(2) * LOG(J(I) / ICOUNT)
160 NEXT I
170 PRINT "The entropy = "; SUM; " bits per symbol."
180 PRINT "A lossless compression of "; 8 / SUM; " to 1 is possible
with entropy coding."
190 CLOSE #1
200 PRINT
210 PRINT "Type any key to end "
220 A$ = INKEY$
230 IF A$ = "" THEN GOTO 220
240 END
```

MSE.BAS

```
10 REM  Program Mean-Square-Error
20 REM this program computes the mean square error between two
```

```
25 REM files
30 n = 1
40 CLASS
50 INPUT "Type filename #1 "; F$
60 INPUT "Type filename #2 "; G$
70 OPEN F$ FOR BINARY AS #1
80 OPEN G$ FOR BINARY AS #2
90 A$ = INPUT$(1, 1)
100 B$ = INPUT$(1, 2)
110 IF EOF(1) THEN GOTO 1000
120 SUM = SUM + (ASC(A$) - ASC(B$)) ^ 2
130 n = n + 1
140 GOTO 90
1000 SUM = SUM / n
1010 PRINT "The mean square error = "; SUM
```

ENTROPY.C

```
#include <math.h>
#include <stdio.h>
/*Huffman Estimator*/
/* Mike Ellis */
main ()
{
char string[80];
unsigned char p;
int i,k;
float j[256],pixel;
float entropy;
FILE *input_file;
FILE *infile;
          entropy = 0;        /*Set all Variable to 0 */
          for ( i = 0; i<=255; i++)
          {
          j[i]=0;
          }
          pixel = 0;
          printf ("\nType name of file ");  /*Open the input file*/
         scanf ("%s",string );
          input_file=fopen(string,"r+b");   /*For Read + Binary */
          while (!feof(input_file))         /*Read to end-of-file*/
         {
          fscanf (input_file,"%c",&p); /*input as unsigned char*/
          pixel = pixel + 1;           /*count characters read*/
          k=p;                         /*convert char to integer*/
          j[k]=j[k]+1;                 /*number of times that */
          }                            /*this char was read */
          for (i = 0; i <=255; i++)    /*compute entropy*/
          {
          if (j[i] !=0)
          {
          entropy = entropy + j[i]*log(j[i]/pixel);
          }
          }
          entropy = -entropy/(pixel*log(2));
          printf ("\nEntropy = %f",entropy); /*print the entropy*/
          printf("\nEntropy encoding can achieve");
          printf(" a %f to 1 compression.\n",8/entropy);
}
```

IMAGE.BAS

```
10 REM
20 REM PROGRAM "IMAGE" FOR DISPLAY GRAYSCALE OR COLOR IMAGES
30 REM USE UNDER QUICKBASIC VERSION 4.5
40 REM
50 CLASS
60 INPUT "Type Image Filename "; C$
70 DEFINT A-Z
80 INPUT "Type X Dimension "; XDIM
90 INPUT "Type Y Dimension "; YDIM
100 GOSUB 210                          'set up 256 color palette
110 OPEN C$ FOR BINARY AS #1
120 WINDOW SCREEN (0, 0)-(XDIM, YDIM)
130 FOR I = 1 TO YDIM
140 A$ = INPUT$(XDIM, 1)
150 FOR J = 1 TO XDIM
160 B = ASC(MID$(A$, J, 1))
170 PSET (J, I),                       'display the pixel
180 NEXT
190 NEXT
200 GOTO 200
210 INPUT "Type Palette Filename "; P$
220 SCREEN 13
230 OPEN P$ FOR BINARY AS #1
240 DIM RED(256), GREEN(256), BLUE(256)
250 FOR K = 0 TO 255
260 A$ = INPUT$(1, 1)
270 RED(K) = ASC(A$)
280 NEXT K
290 FOR K = 0 TO 255
300 A$ = INPUT$(1, 1)
310 GREEN(K) = ASC(A$)
320 NEXT K
330 FOR K = 0 TO 255
340 A$ = INPUT$(1, 1)
350 BLUE(K) = ASC(A$)
360 NEXT K
370 FOR I = 0 TO 255
380 PALETTE I, (INT(BLUE(I) / 4)) * 65536! + 256 * (INT(GREEN(I)/
4)) + INT(RED(I)) / 4
390 NEXT I
400 CLOSE #1
410 RETURN
```

```
10 COMMON DATA1%()                          'More than 64K block
20 CLEAR
30 DIM C(8, 8), DATA1%(90, 1, 8, 8), TEMP(8, 8)
40 DIM TEMP2(8, 8)
50 MAX(1, 1) = -100000!
60 MIN(1, 1) = 100000!
70 REM
80 REM Collect Input Data 8 X 8 Byte Blocks
90 REM
100 INPUT "Type Name of Input File "; F$
110 IF F$ = "" THEN FILES: GOTO 100
120 K = INSTR(1, F$, ".")
130 IF K <> 0 THEN A$ = LEFT$(F$, K - 1) ELSE A$ = F$
140 INPUT "Type X Range "; XINDEX
150 INPUT "Type Y Range "; YINDEX
160 XINDEX = (XINDEX / 8)
170 YINDEX = (YINDEX / 8)
180 K1 = XINDEX * YINDEX
190 G$ = A$ + ".dct"
200 OPEN F$ FOR BINARY AS #1
210 OPEN G$ FOR OUTPUT AS #2
260 GOSUB 870                       'Define cosine transform matrix
290 REM
300 FOR J = 1 TO YINDEX
310 PRINT J * 8
320 FOR Y = 1 TO 8
330 I1 = 0
340 A1$ = INPUT$(8 * XINDEX, 1)
350 FOR I = 1 TO XINDEX
360 FOR X = 1 TO 8
370 I1 = I1 + 1
380 A$ = MID$(A1$, I1, 1)
390 DATA1%(I, 1, Y, X) = ASC(A$) - 128
400 NEXT X
410 NEXT I
420 NEXT Y
430 GOSUB 590                        'Perform the cosine transform
440 FOR X = 1 TO 8
450 FOR I = 1 TO XINDEX
460 FOR Y = 1 TO 8
470 K = (DATA1%(I, 1, X, Y))
480 S$ = S$ + CHR$(K + 128)
490 NEXT
500 NEXT
510 PRINT #2, S$;
520 S$ = ""
530 NEXT
540 REM
550 NEXT
```

```
560 CLOSE #1
570 CLOSE #2
580 END
590 REM
600 REM
610 FOR I = 1 TO XINDEX
620 REM FOR k = 1 TO 8
630 FOR L = 1 TO 8
640 TEMP1 = 0
650 FOR M = 1 TO 8
660 FOR N = 1 TO 8
670 TEMP1 = TEMP1 + DATA1%(I, 1, M, N) * C(L, N)
680 NEXT
690 TEMP(M, L) = TEMP1
700 TEMP1 = 0
710 NEXT
720 NEXT
730 REM NEXT
740 FOR L = 1 TO 8
750 TEMP1 = 0
760 FOR M = 1 TO 8
770 FOR N = 1 TO 8
780 TEMP1 = TEMP1 + C(M, N) * TEMP(N, L)
790 NEXT N
800 DATA1%(I, 1, M, L) = TEMP1 / 8
810 TEMP1 = 0
820 NEXT
830 NEXT
840 REM
850 NEXT I
860 RETURN
870 REM
880 REM Generate Cosine Transform Matrix
890 FOR K = 0 TO 7
900 FOR N = 0 TO 7
910 IF K = 0 THEN C(K + 1, N + 1) = 1 / SQR(8)
920 IF K <> 0 THEN C(K + 1, N + 1) =   SQR(.25) * COS(3.14159 * (2 * N + 1) * K / 16)
930 NEXT N
940 NEXT K
950 RETURN
```

```
10 COMMON DATA1()                          'More than 64K block
20 CLEAR
30 DIM C(8, 8), DATA1(90, 1, 8, 8), TEMP(8, 8)
40 MAX(1, 1) = -100000!
50 MIN(1, 1) = 100000!
60 REM
70 REM Collect Input Data 8 X 8 Byte Blocks
80 REM
90 INPUT "Type Name of Input File "; F$
100 IF F$ = "" THEN FILES: GOTO 90
110 K = INSTR(1, F$, ".")
120 IF K <> 0 THEN A$ = LEFT$(F$, K - 1) ELSE A$ = F$
130 INPUT "Type X Range "; XINDEX
140 INPUT "Type Y Range "; YINDEX
150 XINDEX = (XINDEX / 8)
160 YINDEX = (YINDEX / 8)
170 K1 = XINDEX * YINDEX
180 H$ = A$ + ".rec"
190 OPEN F$ FOR BINARY AS #1
200 OPEN H$ FOR OUTPUT AS #2
250 GOSUB 820                              'Set up C matrix
280 REM
290 FOR J = 1 TO YINDEX
300 PRINT J * 8
310 FOR Y = 1 TO 8
320 FOR I = 1 TO XINDEX
330 FOR X = 1 TO 8
340 A$ = INPUT$(1, 1)
350 DATA1(I, 1, Y, X) = ASC(A$) - 128
360 NEXT X
370 NEXT I
380 NEXT Y
390 GOSUB 550                              'Perform Inverse Cosine
400 REM                                    'Transform
410 FOR X = 1 TO 8
420 FOR I = 1 TO XINDEX
430 FOR Y = 1 TO 8
440 K = CINT(DATA1(I, 1, X, Y))
450 IF K > 127 THEN K = 127
460 IF K < -128 THEN K = -128
470 PRINT #2, CHR$(K + 128);
480 NEXT
490 NEXT
500 NEXT
510 REM
520 NEXT
530 CLOSE #1: CLOSE #2
540 END
550 REM
```

```
560 REM
570 REM
580 FOR I = 1 TO XINDEX
590 FOR L = 1 TO 8
600 TEMP1 = 0
610 FOR M = 1 TO 8
620 FOR N = 1 TO 8
630 TEMP1 = TEMP1 + DATA1(I, 1, M, N) * C(N, L)
640 NEXT
650 TEMP(M, L) = TEMP1
660 TEMP1 = 0
670 NEXT
680 NEXT
690 FOR L = 1 TO 8
700 TEMP1 = 0
710 FOR M = 1 TO 8
720 FOR N = 1 TO 8
730 TEMP1 = TEMP1 + C(N, M) * TEMP(N, L)
740 NEXT N
750 DATA1(I, 1, M, L) = TEMP1 * 8
760 TEMP1 = 0
770 NEXT
780 NEXT
790 REM
800 NEXT
810 RETURN
820 REM
830 REM Generate Cosine Transform Matrix
840 FOR K = 0 TO 7
850 FOR N = 0 TO 7
860 IF K = 0 THEN C(K + 1, N + 1) = 1 / SQR(8)
870 IF K <> 0 THEN C(K + 1, N + 1) =   SQR(.25) * COS(3.14159 * (2
* N + 1) * K / 16)
880 NEXT N
890 NEXT K
900 RETURN
```

IFS.BAS

```
10 KEY OFF
20 INPUT "Type Data File for IFS codes "; IFS$
30 OPEN IFS$ FOR INPUT AS #1
40 I = 1
50 INPUT #1, A(I), B(I), C(I), D(I), E(I), F(I), P(I)
60 P(I) = P(I) + P(I - 1)
70 IF EOF(1) THEN GOTO 100
80 I = I + 1
90 GOTO 50
100 SCREEN 9: CLASS
110 CLOSE #1
120 MINX = 10000: MAXX = -10000: MINY = 10000: MAXY = -10000
130 COLOR 10
140 REM WINDOW (-3, -3)-(5, 15)
150 X = 0: Y = 0: NUMITS = 1000!
160 FOR N = 1 TO NUMITS
170 K1 = INT(RND * 100)
180 FOR J = 1 TO I
190 IF (K1 >= P(J - 1) * 100) AND (K1 < P(J) * 100) THEN K = J
200 NEXT J
210 NEWX = A(K) * X + B(K) * Y + E(K)
220 NEWY = C(K) * X + D(K) * Y + F(K)
230 X = NEWX
240 Y = NEWY
250 OLDK = K
260 IF (N > 10) AND (NUMITS < 10000) THEN GOSUB 330
270 IF (N > 10) AND (NUMITS > 10000) THEN PSET (X, Y)
280 NEXT
290 IF NUMITS > 10000 THEN GOTO 380
300 X = 0: Y = 0: NUMITS = 10000000#
310 WINDOW (MINX - .5 * ABS(MINX), MINY - .5 * ABS(MINY))-(MAXX +
.5 * ABS(MAXX), MAXY + .5 * ABS(MAXY)): CLASS : GOTO 160
320 END
330 IF X > MAXX THEN MAXX = X
340 IF X < MINX THEN MINX = X
350 IF Y > MAXY THEN MAXY = Y
360 IF Y < MINY THEN MINY = Y
370 RETURN
380 A$ = INKEY$
390 IF A$ = "" THEN GOTO 380
400 END
```

IFS Codes for a Square

```
 A   B   C    D    E    F    P
.5 , 0 , 0 , .5 ,  1 ,  1 , .25
.5 , 0 , 0 , .5 , 50 ,  1 , .25
.5 , 0 , 0 , .5 ,  1 , 50 , .25
.5 , 0 , 0 , .5 , 50 , 50 , .25
```

Appendix E
Source Code for "Image Lab" Software

LAB.BAT

```
cl /c /DM5 /AL /I\cscape\include labs.c
cl /c /DM5 /AL /I\cscape\include labx.c
cl /c /DM5 /AL /I\cscape\include laby.c
cl /c /DM5 /AL /I\cscape\include labz.c
link /stack:44000 /SE:300 labs+labx+laby
+labz,,,\global\global+\standard\standard
+\cscape\lib\m1lcscap+\cscape\lib\m1lowl;
```

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <dos.h>
#include <stdarg.h>
#include <ctype.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>
#include "errhand.h"
#include "bitio.h"
#include "\global\globdef.h"
#include "\cscape\include\cscape.h"
#include "\cscape\include\framer.h"

 extern void doinv16(int ixindex, int data16[32][16][16], float c16[16][16]);
 extern void doform16(int ixindex, int data16[32][16][16], float temp16[16][16],
 float c16[16][16]);
 extern void lossy16(int flag1, int data11);

 extern void doinv32(int ixindex, int data32[5][32][32], float c32[32][32]);
 extern void doform32(int ixindex, int data32[5][32][32], float c32[32][32]);
 extern void lossy32(int flag1, int data11);

 extern void analyze(int flag2);

 int vga_code,graphics,width,height,mode,modes[5],choice,
     read_bank,write_bank,top,stat_buf[4],radius,ix,iy,flag16=0,
     colorflag=2;
 char instring1[80],message_string[81],string[80];
 unsigned stringpal1[768];

 sed_type frame;


/*************************************************************************/
/**********************START ADAPTIVE HUFFMAN*****************************/
char *CompressionName1 = "Adaptive Huffman coding, with escape codes";
char *Usage1           = "infile outfile [ -d ]";
#define END_OF_STREAM              256
#define ESCAPE             257
#define SYMBOL_COUNT       258
#define NODE_TABLE_COUNT   ( ( SYMBOL_COUNT * 2 ) - 1 )
#define ROOT_NODE          0
#define MAX_WEIGHT         0x8000
#define TRUE               1
#define FALSE              0
```

```
typedef struct tree {
    int leaf[ SYMBOL_COUNT ];
    int next_free_node;
    struct node {
        unsigned int weight;
        int parent;
        int child_is_leaf;
        int child;
    } nodes[ NODE_TABLE_COUNT ];
} TREE;
TREE Tree;
#ifdef _STDC_
void CompressFile1( FILE *input, BIT_FILE *output );
void ExpandFile1( BIT_FILE *input, FILE *output);
void InitializeTree( TREE *tree );
void EncodeSymbol( TREE *tree, unsigned int c, BIT_FILE *output );
int DecodeSymbol( TREE *tree, BIT_FILE *input );
void UpdateModel( TREE *tree, int c );
void RebuildTree( TREE *tree );
void swap_nodes( TREE *tree, int i, int j );
void add_new_node( TREE *tree, int c );
void PrintTree( TREE *tree );
void print_codes( TREE *tree );
void print_code( TREE *tree, int c );
void calculate_rows( TREE *tree, int node, int level );
int calculate_columns( TREE *tree, int node, int starting_guess );
int find_minimum_column( TREE *tree, int node, int max_row );
void rescale_columns( int factor );
void print_tree( TREE *tree, int first_row, int last_row );
void print_connecting_lines( TREE *tree, int row );
void print_node_numbers( int row );
void print_weights( TREE *tree, int row );
void print_symbol( TREE *tree, int row );
#else
void CompressFile1();
void ExpandFile1();
void InitializeTree();
void EncodeSymbol();
int DecodeSymbol();
void UpdateModel();
void RebuildTree();
void swap_nodes();
void add_new_node();
void PrintTree();
void print_codes();
void print_code();
void calculate_rows();
int calculate_columns();
void rescale_columns();
void print_tree();
void print_connecting_lines();
```

```
void print_node_numbers();
void print_weights();
void print_symbol();
#endif

void CompressFile1( input, output )
FILE *input;
BIT_FILE *output;
{
    int c;

    InitializeTree( &Tree );
    while ( ( c = getc( input ) ) != EOF ) {
        EncodeSymbol( &Tree, c, output );
        UpdateModel( &Tree, c );
    }
    EncodeSymbol( &Tree, END_OF_STREAM, output );
}

void ExpandFile1( input, output )
BIT_FILE *input;
FILE   *output;
{
    int c;

    InitializeTree( &Tree );
    while ( ( c = DecodeSymbol( &Tree, input ) ) != END_OF_STREAM ) {
        if ( putc( c, output ) == EOF )
            fatal_error( "Error writing character" );
        UpdateModel( &Tree, c );
    }
}

void InitializeTree( tree )
TREE *tree;
{
    int i;

    tree->nodes[ ROOT_NODE ].child             = ROOT_NODE + 1;
    tree->nodes[ ROOT_NODE ].child_is_leaf     = FALSE;
    tree->nodes[ ROOT_NODE ].weight            = 2;
    tree->nodes[ ROOT_NODE ].parent            = -1;
    tree->nodes[ ROOT_NODE + 1 ].child         = END_OF_STREAM;
    tree->nodes[ ROOT_NODE + 1 ].child_is_leaf = TRUE;
    tree->nodes[ ROOT_NODE + 1 ].weight        = 1;
    tree->nodes[ ROOT_NODE + 1 ].parent        = ROOT_NODE;
    tree->leaf[ END_OF_STREAM ]                = ROOT_NODE + 1;
    tree->nodes[ ROOT_NODE + 2 ].child         = ESCAPE;
    tree->nodes[ ROOT_NODE + 2 ].child_is_leaf = TRUE;
    tree->nodes[ ROOT_NODE + 2 ].weight        = 1;
    tree->nodes[ ROOT_NODE + 2 ].parent        = ROOT_NODE;
```

```
    tree->leaf[ ESCAPE ]                        = ROOT_NODE + 2;
    tree->next_free_node                        = ROOT_NODE + 3;
    for ( i = 0 ; i < END_OF_STREAM ; i++ )
        tree->leaf[ i ] = -1;
}

void EncodeSymbol( tree, c, output )
TREE *tree;
unsigned int c;
BIT_FILE *output;
{
    unsigned long code;
    unsigned long current_bit;
    int code_size;
    int current_node;

    code = 0;
    current_bit = 1;
    code_size = 0;
    current_node = tree->leaf[ c ];
    if ( current_node == -1 )
        current_node = tree->leaf[ ESCAPE ];
    while ( current_node != ROOT_NODE ) {
        if ( ( current_node & 1 ) == 0 )
            code |= current_bit;
        current_bit <<= 1;
        code_size++;
        current_node = tree->nodes[ current_node ].parent;
    };
    OutputBits( output, code, code_size );
    if ( tree->leaf[ c ] == -1 ) {
        OutputBits( output, (unsigned long) c, 8 );
        add_new_node( tree, c );
    }
}

int DecodeSymbol( tree, input )
TREE *tree;
BIT_FILE *input;
{
    int current_node;
    int c;

    current_node = ROOT_NODE;
    while ( !tree->nodes[ current_node ].child_is_leaf ) {
        current_node = tree->nodes[ current_node ].child;
        current_node += InputBit( input );
    }
    c = tree->nodes[ current_node ].child;
    if ( c == ESCAPE ) {
        c = (int) InputBits( input, 8 );
```

```
        add_new_node( tree, c );
    }
    return( c );
}

void UpdateModel( tree, c )
TREE *tree;
int c;
{
    int current_node;
    int new_node;

    if ( tree->nodes[ ROOT_NODE].weight == MAX_WEIGHT )
        RebuildTree( tree );
    current_node = tree->leaf[ c ];
    while ( current_node != -1 ) {
        tree->nodes[ current_node ].weight++;
        for ( new_node = current_node ; new_node > ROOT_NODE ; new_node- )
            if ( tree->nodes[ new_node - 1 ].weight >=
                 tree->nodes[ current_node ].weight )
                break;
        if ( current_node != new_node ) {
            swap_nodes( tree, current_node, new_node );
            current_node = new_node;
        }
        current_node = tree->nodes[ current_node ].parent;
    }
}

void RebuildTree( tree )
TREE *tree;
{
    int i;
    int j;
    int k;
    unsigned int weight;

    printf( "R" );
    j = tree->next_free_node - 1;
    for ( i = j ; i >= ROOT_NODE ; i- ) {
        if ( tree->nodes[ i ].child_is_leaf ) {
            tree->nodes[ j ] = tree->nodes[ i ];
            tree->nodes[ j ].weight = ( tree->nodes[ j ].weight + 1 ) / 2;
            j-;
        }
    }

    for ( i = tree->next_free_node - 2 ; j >= ROOT_NODE ; i -= 2, j- ) {
        k = i + 1;
        tree->nodes[ j ].weight = tree->nodes[ i ].weight +
                                  tree->nodes[ k ].weight;
```

```
        weight = tree->nodes[ j ].weight;
        tree->nodes[ j ].child_is_leaf = FALSE;
        for ( k = j + 1 ; weight < tree->nodes[ k ].weight ; k++ )
            ;
        k--;
        memmove( &tree->nodes[ j ], &tree->nodes[ j + 1 ],
                ( k - j ) * sizeof( struct node ) );
        tree->nodes[ k ].weight = weight;
        tree->nodes[ k ].child = i;
        tree->nodes[ k ].child_is_leaf = FALSE;
    }
    for ( i = tree->next_free_node - 1 ; i >= ROOT_NODE ; i-- ) {
        if ( tree->nodes[ i ].child_is_leaf ) {
            k = tree->nodes[ i ].child;
            tree->leaf[ k ] = i;
        } else {
            k = tree->nodes[ i ].child;
            tree->nodes[ k ].parent = tree->nodes[ k + 1 ].parent = i;
        }
    }
}


void swap_nodes( tree, i, j )
TREE *tree;
int i;
int j;
{
    struct node temp;

    if ( tree->nodes[ i ].child_is_leaf )
        tree->leaf[ tree->nodes[ i ].child ] = j;
    else {
        tree->nodes[ tree->nodes[ i ].child ].parent = j;
        tree->nodes[ tree->nodes[ i ].child + 1 ].parent = j;
    }
    if ( tree->nodes[ j ].child_is_leaf )
        tree->leaf[ tree->nodes[ j ].child ] = i;
    else {
        tree->nodes[ tree->nodes[ j ].child ].parent = i;
        tree->nodes[ tree->nodes[ j ].child + 1 ].parent = i;
    }
    temp = tree->nodes[ i ];
    tree->nodes[ i ] = tree->nodes[ j ];
    tree->nodes[ i ].parent = temp.parent;
    temp.parent = tree->nodes[ j ].parent;
    tree->nodes[ j ] = temp;
}

void add_new_node( tree, c )
TREE *tree;
```

```
int c;
{
    int lightest_node;
    int new_node;
    int zero_weight_node;

    lightest_node = tree->next_free_node - 1;
    new_node = tree->next_free_node;
    zero_weight_node = tree->next_free_node + 1;
    tree->next_free_node += 2;
    tree->nodes[ new_node ] = tree->nodes[ lightest_node ];
    tree->nodes[ new_node ].parent = lightest_node;
    tree->leaf[ tree->nodes[ new_node ].child ] = new_node;
    tree->nodes[ lightest_node ].child         = new_node;
    tree->nodes[ lightest_node ].child_is_leaf = FALSE;
    tree->nodes[ zero_weight_node ].child          = c;
    tree->nodes[ zero_weight_node ].child_is_leaf  = TRUE;
    tree->nodes[ zero_weight_node ].weight         = 0;
    tree->nodes[ zero_weight_node ].parent         = lightest_node;
    tree->leaf[ c ] = zero_weight_node;
}


struct row {
    int first_member;
    int count;
} rows[ 32 ];


struct location {
    int row;
    int next_member;
    int column;
} positions[ NODE_TABLE_COUNT ];

void PrintTree( tree )
TREE *tree;
{
    int i;
    int min;

    print_codes( tree );
    for ( i = 0 ; i < 32 ; i++ ) {
        rows[ i ].count = 0;
        rows[ i ].first_member = -1;
    }
    calculate_rows( tree, ROOT_NODE, 0 );
    calculate_columns( tree, ROOT_NODE, 0 );
    min = find_minimum_column( tree, ROOT_NODE, 31 );
    rescale_columns( min );
    print_tree( tree, 0, 31 );
```

```
}

void print_codes( tree )
TREE *tree;
{
    int i;

    printf( "\n" );
    for ( i = 0 ; i < SYMBOL_COUNT  ; i++ )
        if ( tree->leaf[ i ] != -1 ) {
             if ( isprint( i ) )
                 printf( "%5c: ", i );
             else
                 printf( "<%3d>: ", i );
            printf( "%5u", tree->nodes[ tree->leaf[ i ] ].weight );
            printf( " " );
            print_code( tree, i );
            printf( "\n" );
        }
}

void print_code( tree, c )
TREE *tree;
int c;
{
    unsigned long code;
    unsigned long current_bit;
    int code_size;
    int current_node;
    int i;

    code = 0;
    current_bit = 1;
    code_size = 0;
    current_node = tree->leaf[ c ];
    while ( current_node != ROOT_NODE ) {
        if ( current_node & 1 )
            code |= current_bit;
        current_bit <<= 1;
        code_size++;
        current_node = tree->nodes[ current_node ].parent;
    };
    for ( i = 0 ; i < code_size ; i++ ) {
        current_bit >>= 1;
        if ( code & current_bit )
            putc( '1', stdout );
        else
            putc( '0', stdout );
    }
}
```

```
void calculate_rows( tree, node, level )
TREE *tree;
int node;
int level;
{
    if ( rows[ level ].first_member == -1 ) {
        rows[ level ].first_member = node;
        rows[ level ].count = 0;
        positions[ node ].row = level;
        positions[ node ].next_member = -1;
    } else {
        positions[ node ].row = level;
        positions[ node ].next_member = rows[ level ].first_member;
        rows[ level ].first_member = node;
        rows[ level ].count++;
    }
    if ( !tree->nodes[ node ].child_is_leaf ) {
        calculate_rows( tree, tree->nodes[ node ].child, level + 1 );
        calculate_rows( tree, tree->nodes[ node ].child + 1, level + 1 );
    }
}

int calculate_columns( tree, node, starting_guess )
TREE *tree;
int node;
int starting_guess;
{
    int next_node;
    int right_side;
    int left_side;

    next_node = positions[ node ].next_member;
    if ( next_node != -1 ) {
        if ( positions[ next_node ].column < ( starting_guess + 4 ) )
            starting_guess = positions[ next_node ].column - 4;
    }
    if ( tree->nodes[ node ].child_is_leaf ) {
        positions[ node ].column = starting_guess;
        return( starting_guess );
    }
    right_side = calculate_columns( tree, tree->nodes[ node ].child,
    starting_guess + 2 );
    left_side = calculate_columns( tree, tree->nodes[ node ].child + 1,
    right_side - 4 );
    starting_guess = ( right_side + left_side ) / 2;
    positions[ node ].column = starting_guess;
    return( starting_guess );
}
int find_minimum_column( tree, node, max_row )
TREE *tree;
int node;
```

```
int max_row;
{
    int min_right;
    int min_left;

    if ( tree->nodes[ node ].child_is_leaf || max_row == 0 )
        return( positions[ node ].column );
    max_row--;
    min_right = find_minimum_column( tree, tree->nodes[ node ].child + 1,
    max_row );
    min_left = find_minimum_column( tree, tree->nodes[ node ].child, max_row );
    if ( min_right < min_left )
        return( min_right );
    else
        return( min_left );
}

void rescale_columns( factor )
int factor;
{
    int i;
    int node;

    for ( i = 0 ; i < 30 ; i++ ) {
        if ( rows[ i ].first_member == -1 )
            break;
        node = rows[ i ].first_member;
        do {
            positions[ node ].column -= factor;
            node = positions[ node ].next_member;
        } while ( node != -1 );
    }


void print_tree( tree, first_row, last_row )
TREE *tree;
int first_row;
int last_row;
{
    int row;

    for ( row = first_row ; row <= last_row ; row++ ) {
        if ( rows[ row ].first_member == -1 )
            break;
        if ( row > first_row )
            print_connecting_lines( tree, row );
        print_node_numbers( row );
        print_weights( tree, row );
        print_symbol( tree, row );
    }
}
```

```
#ifndef ALPHANUMERIC
#define LEFT_END  218
#define RIGHT_END 191
#define CENTER    193
#define LINE      196
#define VERTICAL  179
#else
#define LEFT_END  '+'
#define RIGHT_END '+'
#define CENTER    '+'
#define LINE      '-'
#define VERTICAL  '|'
#endif

void print_connecting_lines( tree, row )
TREE *tree;
int row;

    int current_col;
    int start_col;
    int end_col;
    int center_col;
    int node;
    int parent;

    current_col = 0;
    node = rows[ row ].first_member;
    while ( node != -1 ) {
        start_col = positions[ node ].column + 2;
        node = positions[ node ].next_member;
        end_col = positions[ node ].column + 2;
        parent = tree->nodes[ node ].parent;
        center_col = positions[ parent ].column;
        center_col += 2;
        for ( ; current_col < start_col ; current_col++ )
            putc( ' ', stdout );
        putc( LEFT_END, stdout );
        for ( current_col++ ; current_col < center_col ; current_col++ )
            putc( LINE, stdout );
        putc( CENTER, stdout );
        for ( current_col++; current_col < end_col ; current_col++ )
            putc( LINE, stdout );
        putc( RIGHT_END, stdout );
        current_col++;
        node = positions[ node ].next_member;

    printf( "\n" );


void print_node_numbers( row )
```

```
int row;
{
    int current_col;
    int node;
    int print_col;

    current_col = 0;
    node = rows[ row ].first_member;
    while ( node != -1 ) {
        print_col = positions[ node ].column + 1;
        for ( ; current_col < print_col ; current_col++ )
            putc( ' ', stdout );
        printf( "%03d", node );
        current_col += 3;
        node = positions[ node ].next_member;
    }
    printf( "\n" );
}

void print_weights( tree, row )
TREE *tree;
int row;
{
    int current_col;
    int print_col;
    int node;
    int print_size;
    int next_col;
    char buffer[ 10 ];

    current_col = 0;
    node = rows[ row ].first_member;
    while ( node != -1 ) {
        print_col = positions[ node ].column + 1;
        sprintf( buffer, "%u", tree->nodes[ node ].weight );
        if ( strlen( buffer ) < 3 )
            sprintf( buffer, "%03u", tree->nodes[ node ].weight );
        print_size = 3;
        if ( strlen( buffer ) > 3 ) {
            if ( positions[ node ].next_member == -1 )
                print_size = strlen( buffer );
            else {
                next_col = positions[ positions[ node ].next_member ].column;
                if ( ( next_col - print_col ) > 6 )
                    print_size = strlen( buffer );
                else {
                    strcpy( buffer, "--" );
                    print_size = 3;
                }
            }
        }
```

```
        for ( ; current_col < print_col ; current_col++ )
            putc( ' ', stdout );
        printf( buffer );
        current_col += print_size;
        node = positions[ node ].next_member;
    }
    printf( "\n" );
}


void print_symbol( tree, row )
TREE *tree;
int row;
{
    int current_col;
    int print_col;
    int node;

    current_col = 0;
    node = rows[ row ].first_member;
    while ( node != -1 ) {
        if ( tree->nodes[ node ].child_is_leaf )
            break;
        node = positions[ node ].next_member;
    }
    if ( node == -1 )
        return;
    node = rows[ row ].first_member;
    while ( node != -1 ) {
        print_col = positions[ node ].column + 1;
        for ( ; current_col < print_col ; current_col++ )
            putc( ' ', stdout );
        if ( tree->nodes[ node ].child_is_leaf ) {
            if ( isprint( tree->nodes[ node ].child ) )
                printf( "'%c'", tree-nodes[ node ].child );
            else if ( tree->nodes[ node ].child == END_OF_STREAM )
                printf( "EOF" );
            else if ( tree->nodes[ node ].child == ESCAPE )
                printf( "ESC" );
            else
                printf( "%02XH", tree->nodes[ node ].child );
        } else
            printf( " %c ", VERTICAL );
        current_col += 3;
        node = positions[ node ].next_member;
    }
    printf( "\n" );
}
/************************END ADAPTIVE HUFFMAN*****************************/
/*************************************************************************/
```

```
/*****************************************************************************/
/***************************START HUFFMAN*************************************/
typedef struct tree_node {
    unsigned int count;
    unsigned int saved_count;
    int child_0;
    int child_1;
} NODE;
typedef struct code {
    unsigned int code;
    int code_bits;
} CODE;

#ifdef __STDC__
void count_bytes( FILE *input, unsigned long *long_counts );
void scale_counts( unsigned long *long_counts, NODE *nodes );
int build_tree( NODE *nodes );
void convert_tree_to_code( NODE *nodes,
                           CODE *codes,
                           unsigned int code_so_far,
                           int bits,
                           int node );
void output_counts( BIT_FILE *output, NODE *nodes );
void input_counts( BIT_FILE *input, NODE *nodes );
void print_model( NODE *nodes, CODE *codes );
void compress_data( FILE *input, BIT_FILE *output, CODE *codes );
void expand_data( BIT_FILE *input, FILE *output, NODE *nodes,
                  int root_node );
void print_char( int c );
#else   /* __STDC__ */
void count_bytes();
void scale_counts();
int build_tree();
void convert_tree_to_code();
void output_counts();
void input_counts();
void print_model();
void compress_data();
void expand_data();
void print_char();
#endif  /* __STDC__ */
char *CompressionName2 = "static order 0 model with Huffman coding";
char *Usage2 =
"infile outfile [-d]\n\nSpecifying -d will dump the modeling data\n";

void CompressFile2( input, output )
FILE *input;
BIT_FILE *output;
{
    unsigned long *counts;
    NODE *nodes;
```

```
    CODE *codes;
    int root_node;

    counts = (unsigned long *) calloc( 256, sizeof( unsigned long ) );
    if ( counts == NULL )
        fatal_error( "Error allocating counts array\n" );
    if ( ( nodes = (NODE *) calloc( 514, sizeof( NODE ) ) ) == NULL )
         fatal_error( "Error allocating nodes array\n" );
    if ( ( codes = (CODE *) calloc( 257, sizeof( CODE ) ) ) == NULL )
         fatal_error( "Error allocating codes array\n" );
    count_bytes( input, counts );
    scale_counts( counts, nodes );
    output_counts( output, nodes );
    root_node = build_tree( nodes );
    convert_tree_to_code( nodes, codes, 0, 0, root_node );
    compress_data( input, output, codes );
    free( (char *) counts );
    free( (char *) nodes );
    free( (char *) codes );
}

void ExpandFile2( input, output )
BIT_FILE *input;
FILE *output;
{
    NODE *nodes;
    int root_node;

    if ( ( nodes = (NODE *) calloc( 514, sizeof( NODE ) ) ) == NULL )
        fatal_error( "Error allocating nodes array\n" );
    input_counts( input, nodes );
    root_node = build_tree( nodes );
    expand_data( input, output, nodes, root_node );
    free( (char *) nodes );
}

void output_counts( output, nodes )
BIT_FILE *output;
NODE *nodes;
{
    int first;
    int last;
    int next;
    int i;

    first = 0;
    while ( first < 255 && nodes[ first ].count == 0 )
           first++;
    for ( ; first < 256 ; first = next ) {
        last = first + 1;
        for ( ; ; ) {
```

```
            for ( ; last > 256 ; last++ )
                if ( nodes[ last ].count == 0 )
                    break;
            last--;
            for ( next = last + 1; next < 256 ; next++ )
                if ( nodes[ next ].count != 0 )
                    break;
            if ( next > 255 )
                break;
            if ( ( next - last ) > 3 )
                break;
            last = next;
        };
        if ( putc( first, output->file ) != first )
            fatal_error( "Error writing byte counts\n" );
        if ( putc( last, output->file ) != last )
            fatal_error( "Error writing byte counts\n" );
        for ( i = first ; i <= last ; i++ ) {
            if ( putc( nodes[ i ].count, output->file ) !=
                 (int) nodes[ i ].count )
                fatal_error( "Error writing byte counts\n" );
        }
    }
    if ( putc( 0, output->file ) != 0 )
            fatal_error( "Error writing byte counts\n" );
}


void input_counts( input, nodes )
BIT_FILE *input;
NODE *nodes;
{
    int first;
    int last;
    int i;
    int c;

    for ( i = 0 ; i < 256 ; i++ )
        nodes[ i ].count = 0;
    if ( ( first = getc( input->file ) ) == EOF )
        fatal_error( "Error reading byte counts\n" );
    if ( ( last = getc( input->file ) ) == EOF )
        fatal_error( "Error reading byte counts\n" );
    for ( ; ; ) {
        for ( i = first ; i <= last ; i++ )
            if ( ( c = getc( input->file ) ) == EOF )
                fatal_error( "Error reading byte counts\n" );
            else
                nodes[ i ].count = (unsigned int) c;
        if ( ( first = getc( input->file ) ) == EOF )
            fatal_error( "Error reading byte counts\n" );
```

```
        if ( first == 0 )
            break;
        if ( ( last = getc( input->file ) ) == EOF )
            fatal_error( "Error reading byte counts\n" );
    }
    nodes[ END_OF_STREAM ].count = 1;
}


#ifndef SEEK_SET
#define SEEK_SET 0
#endif

void count_bytes( input, counts )
FILE *input;
unsigned long *counts;
{
    long input_marker;
    int c;

    input_marker = ftell( input );
    while ( ( c = getc( input )) != EOF )
        counts[ c ]++;
    fseek( input, input_marker, SEEK_SET );
}

void scale_counts( counts, nodes )
unsigned long *counts;
NODE *nodes;
{
    unsigned long max_count;
    int i;

    max_count = 0;
    for ( i = 0 ; i < 256 ; i++ )
       if ( counts[ i ] > max_count )
          max_count = counts[ i ];
    if ( max_count == 0 ) {
        counts[ 0 ] = 1;
        max_count = 1;
    }
    max_count = max_count / 255;
    max_count = max_count + 1;
    for ( i = 0 ; i < 256 ; i++ ) {
        nodes[ i ].count = (unsigned int) ( counts[ i ] / max_count );
        if ( nodes[ i ].count == 0 && counts[ i ] != 0 )
            nodes[ i ].count = 1;
    }
    nodes[ END_OF_STREAM ].count = 1;
}
```

```
int build_tree( nodes )
NODE *nodes;
{
    int next_free;
    int i;
    int min_1;
    int min_2;

    nodes[ 513 ].count = 0xffff;
    for ( next_free = END_OF_STREAM + 1 ; ; next_free++ ) {
        min_1 = 513;
        min_2 = 513;
        for ( i = 0 ; i < next_free ; i++ )
            if ( nodes[ i ].count != 0 ) {
                if ( nodes[ i ].count < nodes[ min_1 ].count ) {
                    min_2 = min_1;
                    min_1 = i;
                } else if ( nodes[ i ].count < nodes[ min_2 ].count )
                    min_2 = i;
            }
        if ( min_2 == 513 )
            break;
        nodes[ next_free ].count = nodes[ min_1 ].count
                                 + nodes[ min_2 ].count;
        nodes[ min_1 ].saved_count = nodes[ min_1 ].count;
        nodes[ min_1 ].count = 0;
        nodes[ min_2 ].saved_count =  nodes[ min_2 ].count;
        nodes[ min_2 ].count = 0;
        nodes[ next_free ].child_0 = min_1;
        nodes[ next_free ].child_1 = min_2;
    }
    next_free--;
    nodes[ next_free ].saved_count = nodes[ next_free ].count;
    return( next_free );
}

void convert_tree_to_code( nodes, codes, code_so_far, bits, node )
NODE *nodes;
CODE *codes;
unsigned int code_so_far;
int bits;
int node;
{
    if ( node <= END_OF_STREAM ) {
        codes[ node ].code = code_so_far;
        codes[ node ].code_bits = bits;
        return;
    }
    code_so_far <<= 1;
    bits++;
```

```
    convert_tree_to_code( nodes, codes, code_so_far, bits,
                          nodes[ node ].child_0 );
    convert_tree_to_code( nodes, codes, code_so_far | 1,
                          bits, nodes[ node ].child_1 );
}
void print_model( nodes, codes )
NODE *nodes;
CODE *codes;
{
    int i;

    for ( i = 0 ; i < 513 ; i++ ) {
        if ( nodes[ i ].saved_count != 0 ) {
            printf( "node=" );
            print_char( i );
            printf( "  count=%3d", nodes[ i ].saved_count );
            printf( "  child_0=" );
            print_char( nodes[ i ].child_0 );
            printf( "  child_1=" );
            print_char( nodes[ i ].child_1 );
            if ( codes && i <= END_OF_STREAM ) {
                printf( "  Huffman code=" );
                FilePrintBinary( stdout, codes[ i ].code, codes[ i ].code_bits );
            }
            printf( "\n" );
                }
    }
}

void print_char( c )
int c;
{
    if ( c >= 0x20 && c  127 )
        printf( "'%c'", c );
    else
        printf( "%3d", c );
}

void compress_data( input, output, codes )
FILE *input;
BIT_FILE *output;
CODE *codes;
{
    int c;

    while ( ( c = getc( input ) ) != EOF )
        OutputBits( output, (unsigned long) codes[ c ].code,
                    codes[ c ].code_bits );
    OutputBits( output, (unsigned long) codes[ END_OF_STREAM ].code,
                codes[ END_OF_STREAM ].code_bits );
}
```

```
void expand_data( input, output, nodes, root_node )
BIT_FILE *input;
FILE *output;
NODE *nodes;
int root_node;
{
    int node;

    for ( ; ; ) {
        node = root_node;
        do {
            if ( InputBit( input ) )
                node = nodes[ node ].child_1;
            else
                node = nodes[ node ].child_0;
        } while ( node > END_OF_STREAM );
        if ( node == END_OF_STREAM )
            break;
        if ( ( putc( node, output ) ) != node )
            fatal_error( "Error trying to write expanded byte to output" );
    }
}
/*****************************END HUFFMAN*********************************/
/*************************************************************************/

/*************************************************************************/
/****************************START ARITHMETIC*****************************/
typedef struct {
    unsigned short int low_count;
    unsigned short int high_count;
    unsigned short int scale;
} SYMBOL;
#ifdef __STDC__
void build_model( FILE *input, FILE *output );
void scale_counts3( unsigned long counts[], unsigned char scaled_counts[] );
void build_totals( unsigned char scaled_counts[] );
void count_bytes3( FILE *input, unsigned long counts[] );
void output_counts3( FILE *output, unsigned char scaled_counts[] );
void input_counts3( FILE *stream );
void convert_int_to_symbol( int symbol, SYMBOL *s );
void get_symbol_scale( SYMBOL *s );
int convert_symbol_to_int( int count, SYMBOL *s );
void initialize_arithmetic_encoder( void );
void encode_symbol( BIT_FILE *stream, SYMBOL *s );
void flush_arithmetic_encoder( BIT_FILE *stream );
short int get_current_count( SYMBOL *s );
void initialize_arithmetic_decoder( BIT_FILE *stream );
void remove_symbol_from_stream( BIT_FILE *stream, SYMBOL *s );
#else
void build_model();
```

```
void scale_counts3();
void build_totals();
void count_bytes3();
void output_counts3();
void input_counts3();
void convert_int_to_symbol();
void get_symbol_scale();
int convert_symbol_to_int();
void initialize_arithmetic_encoder();
void encode_symbol();
void flush_arithmetic_encoder();
short int get_current_count();
void initialize_arithmetic_decoder();
void remove_symbol_from_stream();
#endif
short int totals[ 258 ];
char *CompressionName3 = "Fixed order 0 model with arithmetic coding";
char *Usage3           = "in-file out-file\n\n";

void CompressFile3( input, output )
FILE *input;
BIT_FILE *output;
{
    int c;
    SYMBOL s;

    build_model( input, output->file );
    initialize_arithmetic_encoder();
    while ( ( c = getc( input ) ) != EOF ) {
        convert_int_to_symbol( c, &s );
        encode_symbol( output, &s );
    }
    convert_int_to_symbol( END_OF_STREAM, &s );
    encode_symbol( output, &s );
    flush_arithmetic_encoder( output );
    OutputBits( output, 0L, 16 );
}

void ExpandFile3( input, output )
BIT_FILE *input;
FILE *output;
{
    SYMBOL s;
    int c;
    int count;

    input_counts3( input->file );
    initialize_arithmetic_decoder( input );
    for ( ; ; ) {
        get_symbol_scale0( &s );
        count = get_current_count( &s );
```

```
        c = convert_symbol_to_int( count, &s );
        if ( c == END_OF_STREAM )
            break;
        remove_symbol_from_stream( input, &s );
        putc( (char) c, output );
    }
}


void build_model( input, output )
FILE *input;
FILE *output;
{
    unsigned long counts[ 256 ];
    unsigned char scaled_counts[ 256 ];

    count_bytes3( input, counts );
    scale_counts3( counts, scaled_counts );
    output_counts3( output, scaled_counts );
    build_totals( scaled_counts );
}

#ifndef SEEK_SET
#define SEEK_SET 0
#endif

void count_bytes3( input, counts )
FILE *input;
unsigned long counts[];
{
    long input_marker;
    int i;
    int c;

    for ( i = 0 ; i < 256 ; i++ )
        counts[ i ] = 0;
    input_marker = ftell( input );
    while ( ( c = getc( input )) != EOF )
        counts[ c ]++;
    fseek( input, input_marker, SEEK_SET );
}

void scale_counts3( counts, scaled_counts )
unsigned long counts[];
unsigned char scaled_counts[];
{
    int i;
    unsigned long max_count;
    unsigned int total;
    unsigned long scale;
```

```
    max_count = 0;
    for ( i = 0 ; i < 256 ; i++ )
       if ( counts[ i ] > max_count )
          max_count = counts[ i ];
    scale = max_count / 256;
    scale = scale + 1;
    for ( i = 0 ; i < 256 ; i++ ) {
        scaled_counts[ i ] = (unsigned char ) ( counts[ i ] / scale );
        if ( scaled_counts[ i ] == 0 && counts[ i ] != 0 )
            scaled_counts[ i ] = 1;
    }
    total = 1;
    for ( i = 0 ; i < 256 ; i++ )
        total += scaled_counts[ i ];
    if ( total > ( 32767 - 256 ) )
        scale = 4;
    else if ( total > 16383 )
        scale = 2;
    else
        return;
    for ( i = 0 ; i < 256 ; i++ )
        scaled_counts[ i ] /= scale;
}

void build_totals( scaled_counts )
unsigned char scaled_counts[];
{
    int i;

    totals[ 0 ] = 0;
    for ( i = 0 ; i < END_OF_STREAM ; i++ )
        totals[ i + 1 ] = totals[ i ] + scaled_counts[ i ];
    totals[ END_OF_STREAM + 1 ] = totals[ END_OF_STREAM ] + 1;
}

void output_counts3( output, scaled_counts )
FILE *output;
unsigned char scaled_counts[];
{
    int first;
    int last;
    int next;
    int i;

    first = 0;
    while ( first < 255 && scaled_counts[ first ] == 0 )
            first++;
    for ( ; first < 256 ; first = next ) {
        last = first + 1;
        for ( ; ; ) {
            for ( ; last < 256 ; last++ )
```

```
                if ( scaled_counts[ last ] == 0 )
                    break;
            last--;
            for ( next = last + 1; next < 256 ; next++ )
                if ( scaled_counts[ next ] != 0 )
                    break;
            if ( next > 255 )
                break;
            if ( ( next - last ) > 3 )
                break;
            last = next;
        };
        if ( putc( first, output ) != first )
            fatal_error( "Error writing byte counts\n" );
        if ( putc( last, output ) != last )
            fatal_error( "Error writing byte counts\n" );
        for ( i = first ; i <= last ; i++ ) {
            if ( putc( scaled_counts[ i ], output ) !=
                 (int) scaled_counts[ i ] )
                fatal_error( "Error writing byte counts\n" );
        }
    }
    if ( putc( 0, output ) != 0 )
            fatal_error( "Error writing byte counts\n" );
}


void input_counts3( input )
FILE *input;
{
    int first;
    int last;
    int i;
    int c;
    unsigned char scaled_counts[ 256 ];

    for ( i = 0 ; i < 256 ; i++ )
        scaled_counts[ i ] = 0;
    if ( ( first = getc( input ) ) == EOF )
        fatal_error( "Error reading byte counts\n" );
    if ( ( last = getc( input ) ) == EOF )
        fatal_error( "Error reading byte counts\n" );
    for ( ; ; ) {
        for ( i = first ; i <= last ; i++ )
            if ( ( c = getc( input ) ) == EOF )
                fatal_error( "Error reading byte counts\n" );
            else
                scaled_counts[ i ] = (unsigned char) c;
        if ( ( first = getc( input ) ) == EOF )
            fatal_error( "Error reading byte counts\n" );
        if ( first == 0 )
```

```
            break;
        if ( ( last = getc( input ) ) == EOF )
            fatal_error( "Error reading byte counts\n" );
    }
    build_totals( scaled_counts );
}

static unsigned short int code;
static unsigned short int low;
static unsigned short int high;
long underflow_bits;

void initialize_arithmetic_encoder()
{
    low = 0;
    high = 0xffff;
    underflow_bits = 0;
}

void flush_arithmetic_encoder( stream )
BIT_FILE *stream;
{
    OutputBit( stream, low & 0x4000 );
    underflow_bits++;
    while ( underflow_bits-- > 0 )
        OutputBit( stream, ~low & 0x4000 );
}

void convert_int_to_symbol( c, s )
int c;
SYMBOL *s;
{
    s->scale = totals[ END_OF_STREAM + 1 ];
    s->low_count = totals[ c ];
    s->high_count = totals[ c + 1 ];
}

void get_symbol_scale( s )
SYMBOL *s;
{
    s->scale = totals[ END_OF_STREAM + 1 ];
}

int convert_symbol_to_int( count, s)
int count;
SYMBOL *s;
{
    int c;

    for ( c = END_OF_STREAM ; count < totals[ c ] ; c-- )
        ;
```

```
    s->high_count = totals[ c + 1 ];
    s->low_count = totals[ c ];
    return( c );
}

void encode_symbol( stream, s )
BIT_FILE *stream;
SYMBOL *s;
{
    long range;
    range = (long) ( high-low ) + 1;
    high = low + (unsigned short int)
                (( range * s->high_count ) / s->scale - 1 );
    low = low + (unsigned short int)
                (( range * s->low_count ) / s->scale );
    for ( ; ; ) {
        if ( ( high & 0x8000 ) == ( low & 0x8000 ) ) {
            OutputBit( stream, high & 0x8000 );
            while ( underflow_bits > 0 ) {
                OutputBit( stream, ~high & 0x8000 );
                underflow_bits-;
            }
        }
        else if ( ( low & 0x4000 ) && !( high & 0x4000 )) {
            underflow_bits += 1;
            low &= 0x3fff;
            high |= 0x4000;
        } else
            return ;
        low <<= 1;
        high <<= 1;
        high |= 1;
    }
}

short int get_current_count( s )
SYMBOL *s;
{
    long range;
    short int count;

    range = (long) ( high - low ) + 1;
    count = (short int)
            ((((long) ( code - low ) + 1 ) * s->scale-1 ) / range );
    return( count );
}

void initialize_arithmetic_decoder( stream )
BIT_FILE *stream;
{
    int i;
```

```
    code = 0;
    for ( i = 0 ; i < 16 ; i++ ) {
        code <<= 1;
        code += InputBit( stream );
    }
    low = 0;
    high = 0xffff;
}

void remove_symbol_from_stream( stream, s )
BIT_FILE *stream;
SYMBOL *s;
{
    long range;

    range = (long)( high - low ) + 1;
    high = low + (unsigned short int)
                (( range * s->high_count ) / s->scale - 1 );
    low = low + (unsigned short int)
                (( range * s->low_count ) / s->scale );
    for ( ; ; ) {
        if ( ( high & 0x8000 ) == ( low & 0x8000 ) ) {
        }
        else if ((low & 0x4000) == 0x4000  && (high & 0x4000) == 0 ) {
            code ^= 0x4000;
            low   &= 0x3fff;
            high  |= 0x4000;
        } else
            return;
        low <<= 1;
        high <<= 1;
        high |= 1;
        code <<= 1;
        code += InputBit( stream );
    }
}
/*****************************END ARITHMETIC*****************************/
/************************************************************************/

/************************************************************************/
/*******************************START LZW********************************/
#define BITS                        12
#define MAX_CODE                    ( ( 1 << BITS ) - 1 )
#define TABLE_SIZE                  5021
#define FIRST_CODE                  257
#define UNUSED                      -1
#ifdef __STDC__
void CompressFile(FILE *input, BIT_FILE *output);
void ExpandFile( BIT_FILE *input, FILE *output);
unsigned int find_child_node( int parent_code, int child_character );
```

```
unsigned int decode_string( unsigned int offset, unsigned int code );
#else
unsigned int find_child_node();
unsigned int decode_string();
void CompressFile();
void ExpandFile();
#endif
char *CompressionName = "LZW 12 Bit Encoder";
char *Usage           = "in-file out-file\n\n";
struct dictionary {
    int code_value;
    int parent_code;
    char character;
} dict[ TABLE_SIZE ];
char decode_stack[ TABLE_SIZE ];
#define PACIFIER_COUNT 2047

/*****************************START LZWCOM************************/
void CompressFile( input, output)
FILE *input;
BIT_FILE *output;
{
    int next_code;
    int character;
    char character1;
    int string_code;
    unsigned int index;
    unsigned int i;

    next_code = FIRST_CODE;
    for ( i = 0 ; i < TABLE_SIZE ; i++ )
        dict[ i ].code_value = UNUSED;
    if ( ( string_code = getc( input ) ) == EOF )
        string_code = END_OF_STREAM;
    while (( character = getc( input ) ) != EOF) {
        index = find_child_node( string_code, character );
        if ( dict[ index ].code_value != -1 )
            {
            string_code = dict[ index ].code_value;
            }
        else {
            if ( next_code <= MAX_CODE ) {
                dict[ index ].code_value = next_code++;
                dict[ index     rent_code = string_code;
                dict[ inde.     haracter = (char) character;
            }
            OutputBits( output, (unsigned long) string_code, BITS );
            string_code = character;
        }
    }
    OutputBits( output, (unsigned long) string_code, BITS );
```

```
    OutputBits( output, (unsigned long) END_OF_STREAM, BITS );
}
/*****************************END LZWCOM*************************/

/***************************START LZWDECOM***********************/
void ExpandFile( input, output)
BIT_FILE *input;
FILE *output;
{
    unsigned int next_code;
    unsigned int new_code;
    unsigned int old_code;
    int character;
    unsigned int count;

    next_code = FIRST_CODE;
    old_code = (unsigned int) InputBits( input, BITS );
    if ( old_code == END_OF_STREAM )
        return;
    character = old_code;
    putc( old_code, output );
    while ( ( new_code = (unsigned int) InputBits( input, BITS ) )
              != END_OF_STREAM ) {
        if ( new_code >= next_code ) {
            decode_stack[ 0 ] = (char) character;
            count = decode_string( 1, old_code );
        }
        else
            count = decode_string( 0, new_code );
        character = decode_stack[ count - 1 ];
        while ( count > 0 )
            putc( decode_stack[ --count ], output );
        if ( next_code <= MAX_CODE ) {
            dict[ next_code ].parent_code = old_code;
            dict[ next_code ].character = (char) character;
            next_code++;
        }
        old_code = new_code;
    }
}

unsigned int find_child_node( parent_code, child_character )
int parent_code;
int child_character;
{
    int index;
    int offset;
    index = ( child_character << ( BITS - 8 ) ) ^ parent_code;
    if ( index == 0 )
        offset = 1;
    else
```

```
        offset = TABLE_SIZE - index;
    for ( ; ; ) {
        if ( dict[ index ].code_value == UNUSED )
            return( index );
        if ( dict[ index ].parent_code == parent_code &&
             dict[ index ].character == (char) child_character )
            return( index );
        index -= offset;
        if ( index < 0 )
            index += TABLE_SIZE;
    }
}

unsigned int decode_string( count, code )
unsigned int count;
unsigned int code;
{
    while ( code > 255 ) {
        decode_stack[ count++ ] = dict[ code ].character;
        code = dict[ code ].parent_code;
    }
    decode_stack[ count++ ] = (char) code;
    return( count );
}
/*********************************END LZWDECOM*****************************/

/***********************************END LZW*********************************/
/*****************************************************************************/



/************************** START BITIO.C ********************************/
BIT_FILE *OpenOutputBitFile( name )
char *name;
{
    BIT_FILE *bit_file;

    bit_file = (BIT_FILE *) calloc( 1, sizeof( BIT_FILE ) );
    if ( bit_file == NULL )
        return( bit_file );
    bit_file->file = fopen( name, "wb" );
    bit_file->rack = 0;
    bit_file->mask = 0x80;
    bit_file->pacifier_counter = 0;
    return( bit_file );
}

BIT_FILE *OpenInputBitFile( name )
char *name;
{
    BIT_FILE *bit_file;
```

```
    bit_file = (BIT_FILE *) calloc( 1, sizeof( BIT_FILE ) );
    if ( bit_file == NULL )
        return( bit_file );
    bit_file->file = fopen( name, "rb" );
    bit_file->rack = 0;
    bit_file->mask = 0x80;
    bit_file->pacifier_counter = 0;
    return( bit_file );
}

void CloseOutputBitFile( bit_file )
BIT_FILE *bit_file;
{
    if ( bit_file->mask != 0x80 )
        if ( putc( bit_file->rack, bit_file->file ) != bit_file->rack )
            fatal_error( "Fatal error in CloseBitFile!\n" );
    fclose( bit_file->file );
    free( (char *) bit_file );
}

void CloseInputBitFile( bit_file )
BIT_FILE *bit_file;
{
    fclose( bit_file->file );
    free( (char *) bit_file );
}

void OutputBit( bit_file, bit )
BIT_FILE *bit_file;
int bit;
{
    if ( bit )
        bit_file->rack |= bit_file->mask;
    bit_file->mask >>= 1;
    if ( bit_file->mask == 0 ) {
        if ( putc( bit_file->rack, bit_file->file ) != bit_file->rack )
            fatal_error( "Fatal error in OutputBit!\n" );
        else
        if ( ( bit_file->pacifier_counter++ & PACIFIER_COUNT ) == 0 )
                putc( '.', stdout );
        bit_file->rack = 0;
        bit_file->mask = 0x80;
    }
}

void OutputBits( bit_file, code, count )
BIT_FILE *bit_file;
unsigned long code;
int count;
{
```

```
    unsigned long mask;

    mask = 1L << ( count - 1 );
    while ( mask != 0) {
        if ( mask & code )
            bit_file->rack |= bit_file->mask;
        bit_file->mask >>= 1;
        if ( bit_file->mask == 0 ) {
            if ( putc( bit_file->rack, bit_file->file ) != bit_file->rack )
                fatal_error( "Fatal error in OutputBit!\n" );
        else if ( ( bit_file->pacifier_counter++ & PACIFIER_COUNT ) == 0 )
                putc( '.', stdout );
            bit_file->rack = 0;
           bit_file->mask = 0x80;
        }
        mask >>= 1;
    }
}

int InputBit( bit_file )
BIT_FILE *bit_file;
{
    int value;

    if ( bit_file->mask == 0x80 ) {
        bit_file->rack = getc( bit_file->file );
        if ( bit_file->rack == EOF )
            fatal_error( "Fatal error in InputBit!\n" );
    if ( ( bit_file->pacifier_counter++ & PACIFIER_COUNT ) == 0 )
            putc( '.', stdout );
    }
    value = bit_file->rack & bit_file->mask;
    bit_file->mask >>= 1;
    if ( bit_file->mask == 0 )
        bit_file->mask = 0x80;
    return( value ? 1 : 0 );
}

unsigned long InputBits( bit_file, bit_count )
BIT_FILE *bit_file;
int bit_count;
{
    unsigned long mask;
    unsigned long return_value;

    mask = 1L << ( bit_count - 1 );
    return_value = 0;
    while ( mask != 0) {
        if ( bit_file->mask == 0x80 ) {
            bit_file->rack = getc( bit_file->file );
            if ( bit_file->rack == EOF )
```

```
                fatal_error( "Fatal error in InputBit!\n" );
        if ( ( bit_file->pacifier_counter++ & PACIFIER_COUNT ) == 0 )
                putc( '.', stdout );
         }
         if ( bit_file->rack & bit_file->mask )
            return_value |= mask;
        mask >>= 1;
        bit_file->mask >>= 1;
        if ( bit_file->mask == 0 )
            bit_file->mask = 0x80;
    }
    return( return_value );
}

void FilePrintBinary( file, code, bits )
FILE *file;
unsigned int code;
int bits;
{
    unsigned int mask;

    mask = 1 << ( bits - 1 );
    while ( mask != 0 ) {
        if ( code & mask )
            fputc( '1', file );
        else
            fputc( '0', file );
        mask >>= 1;
    }
}
/*************************** END BITIO.C ************************/

/************************ START ERRHAND.C ***********************/
#ifdef __STDC__
void fatal_error( char *fmt, ... )
#else
#ifdef __UNIX__
void fatal_error( fmt, va_alist )
char *fmt;
va_dcl
#else
void fatal_error( fmt )
char *fmt;
#endif
#endif
{
    va_list argptr;
    va_start( argptr, fmt );
    printf( "Fatal error: " );
    vprintf( fmt, argptr );
    va_end( argptr );
```

```
    exit( -1 );
}
/***************************** END ERRHAND.C ************************/

/*****************************************************************************/
/************************COMPRESS/DECOMPRESS*****************************/
int compdecomp(flag,data)
int flag,data;
{
int ch6,ch7;
char string3[80],string4[80];

FILE *input;
BIT_FILE *output;
BIT_FILE *input1;
FILE *output1;

Set_Mode(3);
if (flag == 1) goto decompress;

compress:
if (data == 10) goto huffman;
if (data == 20) goto ahuff;
if (data == 30) goto lzw;
if (data == 40) goto arith;
goto end3;

huffman:
printf ("HUFFMAN COMPRESSION PROGRAM\n");
restart1:
printf("\nType name of input file: ");
scanf("%s",string3);
input = fopen (string3, "r+b");

if (input == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart1;
        }
printf("\nType name of output file: ");
scanf("%s",string4);
printf("\nOutput File = %s\n\n",string4);
output = OpenOutputBitFile(string4);
CompressFile2(input,output);
fclose(input);
CloseOutputBitFile(output);
goto end3;

ahuff:
printf ("ADAPTIVE HUFFMAN COMPRESSION PROGRAM\n");
restart2:
```

```
printf("\nType name of input file: ");
scanf("%s",string3);
input = fopen (string3, "r+b");
if (input == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart2;
        }
printf("\nType name of output file: ");
scanf("%s",string4);
printf("\nOutput File = %s\n\n",string4);
output = OpenOutputBitFile(string4);
CompressFile1(input,output);
fclose(input);
CloseOutputBitFile(output);
goto end3;

lzw:
printf ("LZW COMPRESSION PROGRAM\n");
restart3:
printf("\nType name of input file: ");
scanf("%s",string3);
input = fopen (string3, "r+b");
if (input == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart3;
        }
printf("\nType name of output file: ");
scanf("%s",string4);
printf("\nOutput File = %s\n\n",string4);
output = OpenOutputBitFile(string4);
CompressFile(input,output);
fclose(input);
CloseOutputBitFile(output);
goto end3;

arith:
printf ("ARITHMETIC COMPRESSION PROGRAM\n");
restart4:
printf("\nType name of input file: ");
scanf("%s",string3);
input = fopen (string3, "r+b");
if (input == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart4;
        }
printf("\nType name of output file: ");
scanf("%s",string4);
printf("\nOutput File = %s\n\n",string4);
```

```
output = OpenOutputBitFile(string4);
CompressFile3(input,output);
fclose(input);
CloseOutputBitFile(output);
goto end3;



decompress:
if (data == 10) goto huffman1;
if (data == 20) goto ahuff1;
if (data == 30) goto lzw1;
if (data == 40) goto arith1;
goto end3;

huffman1:
printf ("HUFFMAN DECOMPRESSION PROGRAM\n");
restart5:
printf("\nType name of input file: ");
scanf("%s",string3);
if (input == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart5;
        }
input1 = OpenInputBitFile(string3);
printf("\nType name of output file: ");
scanf("%s",string4);
output1 = fopen (string4, "w+b");
printf("\nOutput File = %s",string4);
printf("\n\n.");
ExpandFile2(input1,output1);
CloseInputBitFile(input1);
fclose(output1);
goto end3;

ahuff1:
printf ("ADAPTIVE HUFFMAN DECOMPRESSION PROGRAM\n");
restart6:
printf("\nType name of input file: ");
scanf("%s",string3);
if (input == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart6;
        }
input1 = OpenInputBitFile(string3);
printf("\nType name of output file: ");
scanf("%s",string4);
output1 = fopen (string4, "w+b");
printf("\nOutput File = %s",string4);
```

```
printf("\n\n.");
ExpandFile1(input1,output1);
CloseInputBitFile(input1);
fclose(output1);
goto end3;

lzw1:
printf ("LZW DECOMPRESSION PROGRAM\n");
restart7:
printf("\nType name of input file: ");
scanf("%s",string3);
input1 = OpenInputBitFile(string3);
if (input == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart7;
        }
printf("\nType name of output file: ");
scanf("%s",string4);
output1 = fopen (string4, "w+b");
printf("\nOutput File = %s\n\n",string4);
ExpandFile(input1,output1);
CloseInputBitFile(input1);
fclose(output1);
goto end3;

arith1:
printf ("ARITHMETIC DECOMPRESSION PROGRAM\n");
restart8:
printf("\nType name of input file: ");
scanf("%s",string3);
input1 = OpenInputBitFile(string3);
if (input == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart8;
        }
printf("\nType name of output file: ");
scanf("%s",string4);
output1 = fopen (string4, "w+b");
printf("\nOutput File = %s\n\n",string4);
ExpandFile3(input1,output1);
CloseInputBitFile(input1);
fclose(output1);

end3:
printf("");
}

/***************************** START DOFORM ***************************/
void doform(ixindex, datal, temp, c)
```

```
int ixindex;
int data1[128][8][8];
float temp[8][8];
float c[8][8];
{
int i,l,m,n;
float temp1;

for ( i = 0; i<ixindex; i++)
        {
        for ( l = 0; l<8; l++)
                {
                temp1 = 0;
                for ( m = 0; m<8; m++)
                        {
                        for ( n = 0; n<8; n++)
                                {
                                temp1 += (float) (data1[i][m][n]) * c[l][n];
                                }
                        temp[m][l] = temp1;
                        temp1 = 0;
                        }
                }
        for ( l = 0; l<8; l++)
                {
                temp1 = 0;
                for (m = 0; m<8; m++)
                        {
                        for ( n = 0; n<8; n++)
                                {
                                temp1 += c[m][n] * temp[n][l];
                                }
                        if (temp1 >= 0)
                                {
                                data1[i][m][l] = (int) (temp1 / 8.0 + 0.5);
                                }
                        else
                                {
                                data1[i][m][l] = (int) (temp1 / 8.0 - 0.5);
                                }
                        temp1 = 0;
                        }
                }
        }
}
/****************************** END DOFORM ***************************/

/****************************** START DOINV ***************************/
void doinv(ixindex, data1, c)

   int ixindex;
```

```
int data1[128][8][8];
float c[8][8];

{
int a,b,q,d;
float temp1, temp[8][8];

 for (a = 0; a<ixindex; a++)
    {
    for (b = 0; b<8; b++)
       {
       temp1 = 0.0;
       for (q = 0; q<8; q++)
          {
          for (d = 0; d<8; d++)
             {
             temp1 += (float) (data1[a][q][d]) * c[d][b];
             }
          temp[q][b] = temp1;
          temp1 = 0.0;
          }
       }
    for (b = 0; b<8; b++)
       {
       temp1 = 0;
       for (q = 0; q<8; q++)
          {
          for (d = 0; d<8; d++)
             {
             temp1 += c[d][q] * temp[d][b];
             }
          if (temp1 >= 0
             {
             data1[a][q][b] = (int) (temp1 * 8 + 0.5);
             }
          else
             {
             data1[a][q][b] = (int) (temp1 * 8 - 0.5);
             }
          temp1 = 0;
          }
       }
    }
}

/****************************** END DOINV ****************************/


/************************** START TRANSFORM ************************/
void transform(flag1,data11)
```

```
int flag1,data11;
{
int xindex,yindex,ixindex,i,j,k,l,n,mm,x,y,data1[128][8][8],
    threshold,width3,height3,xkey,ykey,key,key1,tempr,limit,zone,interm;
float c[8][8],temp[8][8],interm1,interm2;
char string6[80],string7[80];
unsigned char cp1,cp2,cp3,cp4,cp5,cp6;

FILE *input_file1;
FILE *input_file2;
FILE *input_file3;
FILE *output_file1;
FILE *output_file2;
FILE *out1;
FILE *out2;
FILE *out3;

if (flag16 == 1)
        {
        lossy16(flag1,data11);
        goto end1;
        }

if (flag16 == 2)
        {
        lossy32(flag1,data11);
        goto end1;
        }

Set_Mode(3);

if (flag1 == 0) printf("COMPRESSION: ");
if (flag1 == 1) printf("DECOMPRESSION: ");
if (data11 == 15) printf("COSINE-BLACK & WHITE (8 x 8)\n\n");
if (data11 == 18) printf("COSINE-COLOR (8 X 8)\n\n");
if (data11 == 25) printf("HADAMARD-BLACK & WHITE (8 X 8)\n\n");
if (data11 == 28) printf("HADAMARD-COLOR (8 X 8)\n\n");
if (data11 == 35) printf("SINE-BLACK & WHITE (8 X 8)\n\n");
if (data11 == 38) printf("SINE-COLOR (8 X 8)\n\n");

restart9:
printf("Type name of input file: ");
scanf("%s",string6);

if (flag1 == 1)
{
key = 0;
i = 0;
while (string6[i] != '.' && i != 79)
        {
        i = i + 1;
```

```
        if (string6[i] == '.') key = i;
        }
string6[key + 1] = '1';
}
input_file1 = fopen(string6, "r+b");

if (input_file1 == (FILE *) NULL)
        {
        printf("");
        string6[key + 1] = 'd';
        input_file1 = fopen(string6, "r+b");
        }
if (input_file1 == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart9;
        }

printf("\nType name of output file: ");
scanf("%s",string7);

printf("\nEnter width: ");
scanf("%i",&width3);
printf("\nEnter height: ");
scanf("%i",&height3);

if (flag1 == 0)
        {
        printf("\nEnter threshold: ");
        scanf("%i",&threshold);
        restart32:
        printf("\nEnter zonal filter level: ");
        scanf("%i",&zone);
        if (zone > 8 || zone <= 0)
                {
                printf("\nRANGE:  1 - 8!!");
                goto restart32;
                }
        }

if (flag1 == 0)
{
if (data11 == 18 || data11 == 28 || data11 == 38)
        {
        out1 = fopen("Y", "w+b");
        out2 = fopen("Cb","w+b");
        out3 = fopen("Cr","w+b");
        while (!feof(input_file1))
                {
                fscanf (input_file1,"%c",&cp1);
                cp4 = cp1 & 224;
```

```
            interm = cp4;
            interm1 = interm;
            interm2 = interm1 * .299;
            cp5 = (cp1 & 28) * 8;
            interm = cp5;
            interm1 = interm;
            interm2 = interm2 + interm1 * .587;
            cp6 = (cp1 & 3) * 64;
            interm = cp6;
            interm1 = interm;
            interm2 = interm2 + interm1 * .114;
            interm = (interm2 + .5);
            cp6 = interm;
            putc(cp6,out1);
            cp4 = cp1 & 224;
            interm = cp4;
            interm1 = interm;
            interm2 = interm1 * -.16874;
            cp5 = (cp1 & 28) * 8;
            interm = cp5;
            interm1 = interm;
            interm2 = interm2 + interm1 * -.33126;
            cp6 = (cp1 & 3) * 64;
            interm = cp6;
            interm1 = interm;
            interm2 = interm2 + interm1 * .5;
            if (interm2 > 0) interm = (interm2 +.5);
            else interm = (interm2 - .5);
            cp6 = interm + 128;
            putc(cp6,out2);
            cp4 = cp1 & 224;
            interm = cp4;
            interm1 = interm;
            interm2 = interm1 * .5;
            cp5 = (cp1 & 28) * 8;
            interm = cp5;
            interm1 = interm;
            interm2 = interm2 + interm1 * -.41869;
            cp6 = (cp1 & 3) * 64;
            interm = cp6;
            interm1 = interm;
            interm2 = interm2 + interm1 * -.08131;
            if (interm2 > 0) interm = (interm2 +.5);
            else interm = (interm2 - .5);
            cp6 = interm + 128;
            putc(cp6,out3);
            }
        fclose(input_file1);
        fclose(out1);
        fclose(out2);
        fclose(out3);
```

```
        }
}


output_file1 = fopen(string7, "w+b");
if (flag1 == 0)
{
key = 0;
i = 0;
while (string7[i] != '.' && i != 79)
        {
        i = i + 1;
        if (string7[i] == '.') key = i;
        }
string7[key + 1] = '1';
output_file2 = fopen(string7, "w+b");
}
if (data11 == 18 || data11 == 28 || data11 == 38)
        {
        fclose(output_file1);
        fclose(output_file2);
        }


xindex = width3/8;
yindex = height3/8;



if (data11 == 15 || data11 == 18) goto cosinetran;
if (data11 == 25 || data11 == 28) goto hadamard;
if (data11 == 35 || data11 == 38) goto sinetran;


cosinetran:
for ( k = 0; k<=7; k++)
        {
        for ( n = 0; n<=7; n++)
                {
                if (k == 0) c[k][n] = 1 / sqrt(8);
                else c[k][n] = sqrt(.25) * cos(3.14159 * (2 * n + 1) * k / 16);
                }
        }
goto continue1;

hadamard:
c[0][0] = 1; c[0][1] = 1; c[0][2] = 1; c[0][3] = 1;
c[0][4] = 1; c[0][5] = 1; c[0][6] = 1; c[0][7] = 1;
c[1][0] = 1; c[1][1] = -1; c[1][2] = 1; c[1][3] = -1;
c[1][4] = 1; c[1][5] = -1; c[1][6] = 1; c[1][7] = -1;
c[2][0] = 1; c[2][1] = 1; c[2][2] = -1; c[2][3] = -1;
```

```
c[2][4] = 1; c[2][5] = 1; c[2][6] = -1; c[2][7] = -1;
c[3][0] = 1; c[3][1] = -1; c[3][2] = -1; c[3][3] = 1;
c[3][4] = 1; c[3][5] = -1; c[3][6] = -1; c[3][7] = 1;
c[4][0] = 1; c[4][1] = 1; c[4][2] = 1; c[4][3] = 1;
c[4][4] = -1; c[4][5] = -1; c[4][6] = -1; c[4][7] = -1;
c[5][0] = 1; c[5][1] = -1; c[5][2] = 1; c[5][3] = -1;
c[5][4] = -1; c[5][5] = 1; c[5][6] = -1; c[5][7] = 1;
c[6][0] = 1; c[6][1] = 1; c[6][2] = -1; c[6][3] = -1;
c[6][4] = -1; c[6][5] = -1; c[6][6] = 1; c[6][7] = 1;
c[7][0] = 1; c[7][1] = -1; c[7][2] = -1; c[7][3] = 1;
c[7][4] = -1; c[7][5] = 1; c[7][6] = 1; c[7][7] = -1;
for ( i = 0; i<=7; i++)
        {
        for ( j = 0; j<=7; j++)
                {
                c[i][j] = c[i][j] / sqrt(8);
                }
        }
goto continue1;

sinetran:
for (k = 0; k<=7; k++)
        {
        for (n = 0; n<=7; n++)
                {
                c[k][n] = sqrt(.22222222) * sin(3.14159 * (n + 1) * (k + 1)/9);
                }
        }

continue1:
if (data11 == 18 || data11 == 28 || data11 == 38) limit = 3;
else limit = 1;
for (l = 0; l < limit; l++)
{
printf("\n.");
if (8 * yindex != height3) yindex = yindex + 1;
if (flag1 == 0)
        {
        if (data11 == 18 || data11 == 28 || data11 == 38)
                {
                if (l == 0)
                        {
                        input_file1 = fopen("Y", "r+b");
                        key = 0;
                        i = 0;
                        while (string7[i] != '.' && i != 79)
                                {
                                i = i + 1;
                                if (string7[i] == '.') key = i;
                                }
                        string7[key + 1] = 'y';
```

```
                string7[key + 2] = 'y';
                string7[key + 3] = 'y';
                output_file1 = fopen(string7, "w+b");
                string7[key + 1] = '1';
                output_file2 = fopen(string7, "w+b");
                }

        if (l == 1)
                {
                input_file1 = fopen("Cb", "r+b");
                key = 0;
                i = 0;
                while (string7[i] != '.' && i != 79)
                        {
                        i = i + 1;
                        if (string7[i] == '.') key = i;
                        }
                string7[key + 1] = 'c';
                string7[key + 2] = 'c';
                string7[key + 3] = 'b';
                output_file1 = fopen(string7, "w+b");
                string7[key + 1] = '1';
                output_file2 = fopen(string7, "w+b");
                }

        if (l == 2)
                {
                input_file1 = fopen("Cr", "r+b");
                key = 0;
                i = 0;
                while (string7[i] != '.' && i != 79)
                        {
                        i = i + 1;
                        if (string7[i] == '.') key = i;
                        }
                string7[key + 1] = 'c';
                string7[key + 2] = 'c';
                string7[key + 3] = 'r';
                output_file1 = fopen(string7, "w+b");
                string7[key + 1] = '1';
                output_file2 = fopen(string7, "w+b");
                }
        }
    }
if (flag1 == 1)
        {
        if (data11 == 18 || data11 == 28 || data11 == 38)
                {
                if (l == 0)
                        {
                        key = 0;
```

```
                        i = 0;
                        while (string6[i] != '.' && i != 79)
                                {
                                i = i + 1;
                                if (string6[i] == '.') key = i;
                                }
                        string6[key + 1] = '1';
                        string6[key + 2] = 'y';
                        string6[key + 3] = 'y';
                        input_file1 = fopen(string6, "r+b");
                        output_file1 = fopen("Y1", "w+b");
                        }

                if (l == 1)
                        {
                        key = 0;
                        i = 0;
                        while (string6[i] != '.' && i != 79)
                                {
                                i = i + 1;
                                if (string6[i] == '.') key = i;
                                }
                        string6[key + 1] = '1';
                        string6[key + 2] = 'c';
                        string6[key + 3] = 'b';
                        input_file1 = fopen(string6, "r+b");
                        output_file1 = fopen("Cb1", "w+b");
                        }

                if (l == 2)
                        {
                        key = 0;
                        i = 0;
                        while (string6[i] != '.' && i != 79)
                                {
                                i = i + 1;
                                if (string6[i] == '.') key = i;
                                }
                        string6[key + 1] = '1';
                        string6[key + 2] = 'c';
                        string6[key + 3] = 'r';
                        input_file1 = fopen(string6, "r+b");
                        output_file1 = fopen("Cr1", "w+b");
                        }
                }
        }

for( j = 0; j<=yindex-1; j++)
        {
        if (flag1 == 0)
        {
```

```
printf(".");
for ( y = 0; y<=7; y++)
        {
        for ( i = 0; i<=index-1; i++)
                {
                for ( x = 0; x<=7; x++)
                        {
                        if (j * 8 + (y + 1) > height3)
                                {
                                data1[i][y][x] = 0;
                                }
                        else
                                {
                                ykey = y;
                                fscanf(input_file1,"%c",&cp1);
                                data1[i][y][x] = cp1 - 128;
                                }
                        }
                }
                if ( 8 * xindex != width3)
                        {
                        xkey = 8 * xindex;
                        i = xindex;
                        x = 0;
                        while (xkey != width3)
                                {
                                if (j * 8 + (y + 1) > height3)
                                        {
                                        data1[i][y][x] = 0;
                                        }
                                else
                                        {
                                        fscanf(input_file1,"%c",&cp1);
                                        data1[i][y][x] = cp1 - 128;
                                        }
                                x = x + 1;
                                xkey = xkey + 1;
                                }
                        while (x != 8)
                                {
                                data1[i][y][x] = 0;
                                x = x + 1;
                                }
                        }
        }
}
if (flag1 == 1)
{
printf(".");
for ( y = 0; y<=7; y++)
        {
```

```
                if (j * 8 + (y + 1) <= height3) ykey = y;
                for ( i = 0; i<=xindex-1; i++)
                        {
                        for ( x = 0; x<=7; x++)
                                {
                                fscanf(input_file1,"%c",&cp1);
                                data1[i][y][x] = cp1 - 128;
                                }
                        }
                        if ( 8 * xindex != width3)
                                {
                                xkey = 8 * xindex;
                                i = xindex;
                                x = 0;
                                while (xkey != width3)
                                        {
                                        fscanf(input_file1,"%c",&cp1);
                                        data1[i][y][x] = cp1 - 128;
                                        x = x + 1;
                                        xkey = xkey + 1;
                                        }
                                while (x != 8)
                                        {
                                        fscanf(input_file1,"%c",&cp1);
                                        data1[i][y][x] = cp1 - 128;
                                        x = x + 1;
                                        }
                                }
                }
        }

        ixindex = xindex;
        if (8 * xindex != width3) ixindex = ixindex + 1;
        if (flag1 == 0) doform(ixindex, data1, temp, c);
        if (flag1 == 1) doinv(ixindex, data1, c);
        if (flag1 == 0)
        {
        for ( y = 0; y<=7; y++)
                {
                for ( i = 0; i<=xindex-1; i++)
                        {
                        for ( x = 0; x<=7; x++)
                                {
                                if (data1[i][y][x] >= -threshold && data1[i][y][x]
<= threshold) data1[i][y][x] = 0;
                                if (data1[i][y][x] > 127) data1[i][y][x] = 127;
                                if (data1[i][y][x] < -128) data1[i][y][x] = -128;
                                cp1 = data1[i][y][x] + 128;
                                if (j * 8 + (y + 1) <= height3) putc (cp1,output_file1);
                                putc (cp1,output_file2);
                                }
```

```
                        }
                        if ( 8 * xindex != width3)
                                {
                                xkey = 8 * xindex;
                                i = xindex;
                                x = 0;
                                while (xkey != width3)
                                        {
                                        if (data1[i][y][x] >= -threshold && data1[i][y][x]
<= threshold) data1[i][y][x] = 0;
                                        if (data1[i][y][x] > 127) data1[i][y][x] = 127;
                                        if (data1[i][y][x] < -128) data1[i][y][x] = -128;
                                        cp1 = data1[i][y][x] + 128;
                                        if (j * 8 + (y + 1) <= height3) putc (cp1,output_file1);
                                        putc (cp1,output_file2);
                                        x = x + 1;
                                        xkey = xkey + 1;
                                        }
                                while (x != 8)
                                        {
                                        if (data1[i][y][x] >= -threshold && data1[i][y][x]
<= threshold) data1[i][y][x] = 0;
                                        if (data1[i][y][x] > 127) data1[i][y][x] = 127;
                                        if (data1[i][y][x] < -128) data1[i][y][x] = -128;
                                        cp1 = data1[i][y][x] + 128;
                                        putc (cp1,output_file2);
                                        x = x + 1;
                                        }
                                }
                        }
                }
                if (flag1 == 1)
                {
                for ( y = 0; y<=ykey; y++)
                        {
                        for ( i = 0; i<=xindex-1; i++)
                                {
                                for ( x = 0; x<=7; x++)
                                        {
                                        if (data1[i][y][x] > 127) data1[i][y][x] = 127;
                                        if (data1[i][y][x] < -128) data1[i][y][x] = -128;
                                        cp1 = data1[i][y][x] + 128;
                                        putc (cp1,output_file1);
                                        }
                                }
                                if ( 8 * xindex != width3)
                                        {
                                        xkey = 8 * xindex;
                                        i = xindex;
                                        x = 0;
                                        while (xkey != width3)
```

```
                                        {
                                        if (data1[i][y][x] > 127) data1[i][y][x] = 127;
                                        if (data1[i][y][x] < -128) data1[i][y][x] = -128;
                                        cp1 = data1[i][y][x] + 128;
                                        putc (cp1,output_file1);
                                        x = x + 1;
                                        xkey = xkey + 1;
                                        }
                                }
                        }
                }
                }
fclose(input_file1);
fclose(output_file1);
fclose(output_file2);
}

if (zone == 8) goto jump1;
if (flag1 == 0)
{
input_file1 = fopen(string7, "r+b");
output_file1 = fopen("temp1", "w+b");

if (xindex*8 != width3) xindex = xindex + 1;

for (y = 0; y < yindex; y++)
        {
        for (i = 0; i < 8; i++)
        {
        for (x = 0; x < xindex; x++)
                {
                for (j = 0; j < 8; j++)
                {
                fscanf(input_file1,"%c",&cp1);
                putc(cp1,output_file1);
                }
                }
        }
        }

fclose(input_file1);
fclose(output_file1);

input_file1 = fopen("temp1", "r+b");
output_file1 = fopen(string7, "w+b");

for (y = 0; y < yindex; y++)
        {
        for (i = 0; i < 8; i++)
        {
        for (x = 0; x < xindex; x++)
```

```
                {
                for (j = 0; j < 8; j++)
                {
                fscanf(input_file1,"%c",&cp1);
                if (i >= zone || j >=zone) cp1 = 128;
                putc(cp1,output_file1);
                }
                }
        }
        }

fclose(input_file1);
fclose(output_file1);
}
jump1:
if (data11 == 18 || data11 == 28 || data11 == 38)
        {
        if (flag1 == 1)
        {
        input_file1 = fopen("Y1", "r+b");
        input_file2 = fopen("Cb1", "r+b");
        input_file3 = fopen("Cr1", "r+b");
        output_file1 = fopen(string7, "w+b");
        while (!feof(input_file1) && !feof(input_file2) && !feof(input_file3))
                {
                fscanf (input_file1,"%c",&cp1);
                fscanf (input_file2,"%c",&cp2);
                fscanf (input_file3,"%c",&cp3);
                interm = cp1;
                interm2 = interm;
                interm = cp3 - 128;
                interm1 = interm;
                interm2 = interm2 + interm1 * 1.402;
                interm = (interm2 + .5);
                if (interm < 0) interm = 0;
                cp4 = interm;
                if (cp4 > 208) cp4 = 224;
                else if (cp4 > 176) cp4 = 192;
                        else if (cp4 > 144) cp4 = 160;
                                else if (cp4 > 112) cp4 = 128;
                                        else if (cp4 > 80) cp4 = 96;
                                                else if (cp4 > 48) cp4 = 64;
                                                        else if (cp4 > 16) cp4 = 32;
                                                                else cp4 = 0;
                interm = cp2;
                interm2 = interm;
                interm = cp2 - 128;
                interm1 = interm;
                interm1 = interm1 * .34414;
                interm2 = interm2 - interm1;
                interm = cp3 - 128;
```

```
                interm1 = interm;
                interm1 = interm1 * .71414;
                interm2 = interm2 - interm1;
                interm = (interm2 + .5);
                if (interm < 0) interm = 0;
                cp5 = interm;
                if (cp5 > 208) cp5 = 224;
                else if (cp5 > 176) cp5 = 192;
                        else if (cp5 > 144) cp5 = 160;
                                else if (cp5 > 112) cp5 = 128;
                                        else if (cp5 > 80) cp5 = 96;
                                                else if (cp5 > 48) cp5 = 64;
                                                        else if (cp5 > 16) cp5 = 32;
                                                                else cp5 = 0;
                cp5 = cp5 / 8;
                interm = cp1;
                interm2 = interm;
                interm = cp2 - 128;
                interm1 = interm;
                interm2 = interm2 + interm1 * 1.772;
                interm = (interm2 + .5);
                if (interm < 0) interm = 0;
                cp6 = interm;
                if (cp6 > 160) cp6 = 192;
                else if (cp6 > 96) cp6 = 128;
                        else if (cp6 > 32) cp6 = 64;
                                else cp6 = 0;
                cp6 = cp6 / 64;
                cp1 = cp4 + cp5 + cp6;
                putc (cp1,output_file1);
                }
        fclose(input_file1);
        fclose(input_file2);
        fclose(input_file3);
        fclose(output_file1);
        remove("Y1");
        remove("Cb1");
        remove("Cr1");
        }
        if (flag1 == 0)
        {
        remove("Y");
        remove("Cb");
        remove("Cr");
        }
        }

endl:
printf("");
}
/*************************** END TRANSFORM ************************/
```

```
/************************COMPRESS/DECOMPRESS*****************************/
/************************************************************************/

/************************VIDEO MODES*********************/
void graphics1(idata, mask, ch)
int idata,mask,ch;
{
int i,j,p,l,m,ii,jj,kk,mm,red2[8],green2[8],blue2[8],x,y,ch3;
unsigned char red[256],green[256],blue[256],red1[256],green1[256],
    blue1[256],point_color,cp;

FILE *input_file;
FILE *inpalette;
FILE *out;

if (idata == 21) goto loadimage;
if (idata == 22) goto loadpalette;
if (ch != 0) goto image;
if (idata == 35) goto setmode;
if (idata == 36) goto makepalette;

setmode:
Set_Mode(3);
Build_Mode();
vga_code=Which_VGA();
VGA_id = vga_code;
if(vga_code<0)
        {
        SetMode(3);
        strcpy(message_string,"ERROR Exit 100:  Can't Detect VGA");
        Message_On_Screen(message_string);
        exit(1);
        }
graphics = 1;
Number_Color = 256;
for ( i = 1; i<=4; i++)
        {
        modes[i] = 0;
        }
width = 1024;
height = 768;
mode = Find_Mode(width,height,Number_Color,graphics);
if (mode != -1) modes[4] = mode;
width = 800;
height = 600;
mode = Find_Mode(width,height,Number_Color,graphics);
if (mode != -1) modes[3] = mode;
width = 640;
height = 480;
mode = Find_Mode(width,height,Number_Color,graphics);
```

```
if (mode != -1) modes[2] = mode;
width = 320;
height = 200;
mode = Find_Mode(width,height,Number_Color,graphics);
if (mode != -1) modes[1] = mode;
printf("AVAILABLE MODES\n");
if (modes[1] != 0) printf("\n1.  320X200");
if (modes[2] != 0) printf("\n2.  640X480");
if (modes[3] != 0) printf("\n3.  800X600");
if (modes[4] != 0) printf("\n4.  1024X768");
printf("\n\nSelection: ");
ch3 = getch();
printf("%c",ch3);
if (ch3 == '1')
        {
        width = 320;
        mode = modes[1];
        }
if (ch3 == '2')
        {
        width = 640;
        mode = modes[2];
        }
if (ch3 == '3')
        {
        width = 800;
        mode = modes[3];
        }
if (ch3 == '4')
        {
        width = 1024;
        mode = modes[4];
        }
if (mode == -1)
        printf ("\nError");
if(Number_Color==256)
 { Load_Write_Bank_256(0); Load_Read_Bank_256(0); }
if(Number_Color==16)
 { Load_Write_Bank_16(0);  Load_Read_Bank_16(0);  }
getch();
goto end4;
/************************VIDEO MODES*********************/

/************************LOAD PALETTE********************/
loadpalette:
Set_Mode(3);
restart18:
printf ("Type name of palette file: ");
scanf ("%s",instring1);
inpalette=fopen(instring1,"r+b");
if (inpalette == (FILE *) NULL)
```

```
        {
          printf("\n Bad file - try again  \n \n");
          goto restart18;
        }
p = 0;
while (!feof(inpalette))
        {
        fscanf (inpalette,"%c",&stringpal1[p]);
        p = p + 1;
        }
for ( ii = 0; ii<=255; ii++)
        {
        red[ii] = stringpal1[ii];
        }
for ( jj = 256; jj<=511; jj++)
        {
        l = jj - 256;
        green[l] = stringpal1[jj];
        }
for ( kk = 512; kk<=767; kk++)
        {
        m = kk - 512;
        blue[m] = stringpal1[kk];
        }
outp(0x3c8,0);
for ( mm = 0; mm<=255; mm++)
         {
         if (ch == 0 || ch == 1 || ch == 4 || ch == 6) outp(0x3c9,red[mm]/4); else outp(0x3c9,0);
         if (ch == 0 || ch == 2 || ch == 4 || ch == 6) outp(0x3c9,green[mm]/4); else outp(0x3c9,0);
         if (ch == 0 || ch == 3 || ch == 4 || ch == 6) outp(0x3c9,blue[mm]/4); else outp(0x3c9,0);
         }

goto end4;
/************************LOAD PALETTE*********************/

/************************MAKE PALETTE*********************/
makepalette:
Set_Mode(3);
printf("Enter name of output file: ");
scanf ("%s",instring1);
out = fopen(instring1,"w+b");
printf("\nEnter 8 red intensities in ascending order:\n");
for ( i = 0; i<=7; i++)
        {
        scanf("%i",&red2[i]);
        red1[i] = red2[i];
        }
printf("Enter 8 green intensities in ascending order:\n");
for ( i = 0; i<=7; i++)
        {
        scanf("%i",&green2[i]);
```

```
        green1[i] = green2[i];
        }
printf("Enter 4 blue intensities in ascending order:\n");
for ( i = 0; i<=3; i++)
        {
        scanf("%i",&blue2[i]);
        blue1[i] = blue2[i];
        }
for ( i = 0; i<=31; i++)
        {
        stringpal1[i] = red1[0];
        }
for ( i = 32; i<=63; i++)
        {
        stringpal1[i] = red1[1];
        }
for ( i = 64; i<=95; i++)
        {
        stringpal1[i] = red1[2];
        }
for ( i = 96; i<=127; i++)
        {
        stringpal1[i] = red1[3];
        }
for ( i = 128; i<=159; i++)
        {
        stringpal1[i] = red1[4];
        }
for ( i = 160; i<=191; i++)
        {
        stringpal1[i] = red1[5];
        }
for ( i = 192; i<=223; i++)
        {
        stringpal1[i] = red1[6];
        }
for ( i = 224; i<=255; i++)
        {
        stringpal1[i] = red1[7];
        }
for ( i = 0; i<=7; i++)
        {
        for ( j = 0; j<=3; j++)
                {
                stringpal1[32 * i + j + 256] = green1[0];
                }
        for ( j = 4; j<=7; j++)
                {
                stringpal1[32 * i + j + 256] = green1[1];
                }
        for ( j = 8; j<=11; j++)
```

```
                {
                stringpall[32 * i + j + 256] = green1[2];
                }
        for ( j = 12; j<=15; j++)
                {
                stringpall[32 * i + j + 256] = green1[3];
                }
        for ( j = 16; j<=19; j++)
                {
                stringpall[32 * i + j + 256] = green1[4];
                }
        for ( j = 20; j<=23; j++)
                {
                stringpall[32 * i + j + 256] = green1[5];
                }
        for ( j = 24; j<=27; j++)
                {
                stringpall[32 * i + j + 256] = green1[6];
                }
        for ( j = 28; j<=31; j++)
                {
                stringpall[32 * i + j + 256] = green1[7];
                }
        }
for ( i = 0; i<=63; i++)
        {
        stringpall[4 * i + 512] = blue1[0];
        stringpall[4 * i + 1 + 512] = blue1[1];
        stringpall[4 * i + 2 + 512] = blue1[2];
        stringpall[4 * i + 3 + 512] = blue1[3];
        }
for ( i = 0; i<=767; i++)
        {
        putc (stringpall[i], out);
        }
fclose (out);
goto end4;
/************************MAKE PALETTE*********************/


/*****************************IMAGE*************************/
loadimage:
Set_Mode(3);
restart19:
printf ("Type name of image file: ");
scanf ("%s",string);
input_file=fopen(string,"r+b");
if (input_file == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart19;
```

```
        }
printf ("\nType X dimension: ");
scanf ("%i",&ix);
printf ("\nType Y dimension: ");
scanf ("%i",&iy);

image:
Set_Mode(3);
if (ch != 0) input_file=fopen(string,"r+b");
Set_Mode(mode);
for ( ii = 0; ii<=255; ii++)
        {
        red[ii] = stringpal1[ii];
        }
for ( jj = 256; jj<=511; jj++)
        {
        l = jj - 256;
        green[l] = stringpal1[jj];
        }
for ( kk = 512; kk<=767; kk++)
        {
        m = kk - 512;
        blue[m] = stringpal1[kk];
        }
outp(0x3c8,0);
for ( mm = 0; mm<=255; mm++)
         {
         if (ch == 0 || ch == 1 || ch == 4 || ch == 6) outp(0x3c9,red[mm]/4); else outp(0x3c9,0);
         if (ch == 0 || ch == 2 || ch == 4 || ch == 6) outp(0x3c9,green[mm]/4); else outp(0x3c9,0);
         if (ch == 0 || ch == 3 || ch == 4 || ch == 6) outp(0x3c9,blue[mm]/4); else outp(0x3c9,0);
         }
for (y=0; y<=y-1;y++)
        {
        for (x=0; x<=ix-1;x++)
               {
               if (y == 200 && width == 320) goto end9;
               if (y == 480 && width == 640) goto end9;
               if (y == 600 && width == 800) goto end9;
               if (y == 768 && width == 1024) goto end9;
               fscanf (input_file, "%c",&cp);
               if (ch == 4)
                   {
                   point_color = cp & mask;
                   if (point_color && ch == 4) point_color = 255;
                   }
                else point_color = cp;
                if (x < 1024) WrPixel_256(x,y,point_color,width);
                }
        }
        end9:
        fclose(input_file);
```

```
        getch();

end4:
printf("");
}
/***************************IMAGE************************/

/***************************FILE*************************/

void copyf(void)
{
char stringa[20],stringb[20];
unsigned char cp1;

FILE *input;
FILE *output;

Set_Mode(3);
restart40:
printf("Type old file name: ");
scanf("%s",stringa);
input = fopen(stringa, "r+b");
if (input == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart40;
        }
printf("\nType new file name: ");
scanf("%s",stringb);
output = fopen(stringb, "w+b");

while (!feof(input))
        {
        fscanf(input,"%c",&cp1);
        putc(cp1,output);
        }
fclose(input);
fclose(output);
}

void deletef(void)
{
char stringa[20];
unsigned char cp1;

FILE *input;

Set_Mode(3);
printf("Type file name to delete: ");
scanf("%s",stringa);
remove(stringa);
```

```
fclose(input);
}

void rname(void)
{
char stringa[20],stringb[20];
unsigned char cp1;

FILE *input;
FILE *output;

Set_Mode(3);
restart41:
printf("Type old file name: ");
scanf("%s",stringa);
input = fopen(stringa, "r+b");
if (input == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart41;
        }
printf("\nType new file name: ");
scanf("%s",stringb);
output = fopen(stringb, "w+b");

while (!feof(input))
        {
        fscanf(input,"%c",&cp1);
        putc(cp1,output);
        }
remove(stringa);
fclose(input);
fclose(output);
}

/***************************FILE*************************/

int metro(sdata, data)
char *sdata;
int data;
{
int mask,ch;

mask = 255;
ch = 0;
if (data == 23) ch = 1;
if (data == 24) ch = 2;
if (data == 25) ch = 3;
if (data == 26)
        {
        ch = 4;
```

```
        mask = 128;
        }
if (data == 27)
        {
        ch = 4;
        mask = 64;
        }
if (data == 28)
        {
        ch = 4;
        mask = 32;
        }
if (data == 29)
        {
        ch = 4;
        mask = 16;
        }
if (data == 30)
        {
        ch = 4;
        mask = 8;
        }
if (data == 31)
        {
        ch = 4;
        mask = 4;
        }
if (data == 32)
        {
        ch = 4;
        mask = 2;
        }
if (data == 33)
        {
        ch = 4;
        mask = 1;
        }
if (data == 34) ch = 6;
graphics1(data,mask,ch);
Set_Mode(3);
return(1);
}

int file(sdata, data)
char *sdata;
int data;
{
if (data == 1) copyf();
if (data == 2) deletef();
if (data == 3) rname();
Set_Mode(3);
```

```
return(1);
}

void getout(idata)
int idata;
{
Set_Mode(3);
exit(1);
}

sed_type frame1;

int comp12(sdata, idata)
char *sdata;
int idata;
{
compdecomp(0,idata);
Set_Mode(3);
return(1);
}

struct frame_def lossless_frame[] = {

{ "HUFFMAN-0  ", comp12, 10},
{ FRAME_END },

{ "ADAPTIVE HUFFMAN  ", comp12, 20},
{ FRAME_END },

{ "LZW  ", comp12, 30},
{ FRAME_END },

{ "ARITHMETIC", comp12, 40},
{ FRAME_END },
{ FRAME_END }
};

int comp1(idata)
int idata;
{
VOID *sdata;

Set_Mode(3);
if (colorflag == 0) frame1 = frame_Open(lossless_frame, bd_1, 0x04, 0x47, 0x07);
if (colorflag == 1) frame1 = frame_Open(lossless_frame, bd_1, 0x02, 0x7A, 0x07);
if (colorflag == 2) frame1 = frame_Open(lossless_frame, bd_1, 0x01, 0x7B, 0x07);
if (colorflag == 3) frame1 = frame_Open(lossless_frame, bd_1, 0x07, 0x70, 0x07);
frame_Repaint(frame1);
frame_Go(frame1, ' ', (VOID *) sdata);
frame_Close(frame1);
}
```

```
sed_type frame2;

int decomp12(sdata, idata)
char *sdata;
int idata;
{
compdecomp(1,idata);
Set_Mode(3);
return(1);
}

struct frame_def lossless_frame1[] = {

{ "HUFFMAN-0  ", decomp12, 10},
{ FRAME_END },

{ "ADAPTIVE HUFFMAN  ", decomp12, 20},
{ FRAME_END },

{ "LZW  ", decomp12, 30},
{ FRAME_END },
{ "ARITHMETIC", decomp12, 40},
{ FRAME_END },
{ FRAME_END }
};

int decomp1(idata)
int idata;
{
VOID *sdata;

Set_Mode(3);
if (colorflag == 0) frame2 = frame_Open(lossless_frame1, bd_1, 0x04, 0x47, 0x07);
if (colorflag == 1) frame2 = frame_Open(lossless_frame1, bd_1, 0x02, 0x7A, 0x07);
if (colorflag == 2) frame2 = frame_Open(lossless_frame1, bd_1, 0x01, 0x7B, 0x07);
if (colorflag == 3) frame2 = frame_Open(lossless_frame1, bd_1, 0x07, 0x70, 0x07);
frame_Repaint(frame2);
frame_Go(frame2, ' ', (VOID *) sdata);
frame_Close(frame2);
}

sed_type frame3;

int comp22(sdata, idata)
char *sdata;
int idata;
{
transform(0,idata);
Set_Mode(3);
return(1);
```

```
}

struct frame_def lossy_frame[] = {

{ "COSINE  ", NULL, 10},

{ "Black & White", comp22, 15},
{ "Color", comp22, 18},
{ FRAME_END },

{ "HADAMARD  ", NULL, 20},

{ "Black & White", comp22, 25},
{ "Color", comp22, 28},
{ FRAME_END },

{ "SINE  ", NULL, 30},

{ "Black & White", comp22, 35},
{ "Color", comp22, 38},
{ FRAME_END },
{ FRAME_END }
};

int comp2(idata)
int idata;
{
VOID *sdata;

Set_Mode(3);
if (colorflag == 0) frame3 = frame_Open(lossy_frame, bd_1, 0x04, 0x47, 0x07);
if (colorflag == 1) frame3 = frame_Open(lossy_frame, bd_1, 0x02, 0x7A, 0x07);
if (colorflag == 2) frame3 = frame_Open(lossy_frame, bd_1, 0x01, 0x7B, 0x07);
if (colorflag == 3) frame3 = frame_Open(lossy_frame, bd_1, 0x07, 0x70, 0x07);
frame_Repaint(frame3);
frame_Go(frame3, ' ', (VOID *) sdata);
frame_Close(frame3);
}

sed_type frame4;

int decomp22(sdata, idata)
char *sdata;
int idata;
{
transform(1,idata);
Set_Mode(3);
return(1);
}

struct frame_def lossy_frame1[] = {
```

```
{ "COSINE  ", NULL, 10},

{ "Black & White", deccmp22, 15},
{ "Color", decomp22, 18},
{ FRAME_END },

{ "HADAMARD  ", NULL, 20},

{ "Black & White", decomp22, 25},
{ "Color", decomp22, 28},
{ FRAME_END },

{ "SINE  ", NULL, 30},

{ "Black & White", decomp22, 35},
{ "Color", decomp22, 38},
{ FRAME_END },
{ FRAME_END }

};

int decomp2(idata)
int idata;
{
VOID *sdata;

Set_Mode(3);
if (colorflag == 0) frame4 = frame Open(lossy_frame1, bd_1, 0x04, 0x47, 0x07);
if (colorflag == 1) frame4 = frame_Open(lossy_frame1, bd_1, 0x02, 0x7A, 0x07);
if (colorflag == 2) frame4 = frame_Open(lossy_frame1, bd_1, 0x01, 0x7B, 0x07);
if (colorflag == 3) frame4 = frame_Open(lossy_frame1, bd_1, 0x07, 0x70, 0x07);
frame_Repaint(frame4);
frame_Go(frame4, ' ', (VOID *) sdata);
frame_Close(frame4);
}

int analysis0(idata)
int idata;
{
analyze(0);
Set_Mode(3);
return(1);
}

int analysis1(idata)
int idata;
{
analyze(1);
Set_Mode(3);
return(1);
```

```
}

int analysis2(idata)
int idata;
{
analyze(2);
Set_Mode(3);
return(1);
}

int analysis3(idata)
int idata;
{
analyze(3);
Set_Mode(3);
return(1);
}

int analysis4(idata)
int idata;
{
analyze(4);
Set_Mode(3);
return(1);
}

sed_type frame5;

int pasteh(idata)
int idata;
{
analyze(6);
Set_Mode(3);
return(1);
}

int pastev(idata)
int idata;
{
analyze(7);
Set_Mode(3);
return(1);
}

struct frame_def chop_frame1[] = {

{ "HORIZONTALLY  ", pasteh, 10},
{ FRAME_END },

{ "VERTICALLY  ", pastev, 20},
{ FRAME_END },
```

```
{ FRAME_END }
};

int analysis5(idata)
int idata;
{
analyze(5);
Set_Mode(3);
return(1);
}

int analysis6(idata)
int idata;
{
Set_Mode(3);
if (colorflag == 0) frame5 = frame_Open(chop_frame1, bd_1, 0x04, 0x47, 0x07);
if (colorflag == 1) frame5 = frame_Open(chop_frame1, bd_1, 0x02, 0x7A, 0x07);
if (colorflag == 2) frame5 = frame_Open(chop_frame1, bd_1, 0x01, 0x7B, 0x07);
if (colorflag == 3) frame5 = frame_Open(chop_frame1, bd_1, 0x07, 0x70, 0x07);
frame_Repaint(frame5);
frame_Go(frame5, ' ', NULL);
frame_Close(frame5);
Set_Mode(3);
return(1);
}

int analysis7(idata)
int idata;
{
analyze(8);
Set_Mode(3);
return(1);
}


int lmode(idata)
int idata;
{
int ch16;

Set_Mode(3);
printf("AVAILABLE LOSSY MODES\n");
printf("\n1.  8 X 8");
printf("\n2.  16 X 16");
printf("\n3.  32 X 32");
printf("\n\nMode Selection: ");
ch16 = getch();
printf("%c",ch16);
getch();
if (ch16 == '1') flag16 = 0;
if (ch16 == '2') flag16 = 1;
```

```
if (ch16 == '3') flag16 = 2;

if (ch16 == '3')
        {
        printf("\n\nNOTE: This mode supports a maximum width of 640.");
        getch();
        }

Set_Mode(3);
return(1);
}

int mmode(idata)
int idata;
{
int ch30;

Set_Mode(3);
printf("AVAILABLE MENU COLORS\n");
printf("\n1.  Red");
printf("\n2.  Green");
printf("\n3.  Blue");
printf("\n4.  B&W");
printf("\n\nSelection: ");
ch30 = getch();
printf("%c",ch30);
getch();
if (ch30 == '1') colorflag = 0;
if (ch30 == '2') colorflag = 1;
if (ch30 == '3') colorflag = 2;
if (ch30 == '4') colorflag = 3;
Set_Mode(3);
return(1);
}


 static struct frame_def main_frame[] = {

 { "FILE  ", NULL, 10},

 { "Copy File", file, 1},
 { "Delete File", file, 2},
 { "Load File", metro, 21},
 { "Load Palette", metro, 22},
 { "Rename File", file, 3},
 { "Quit", getout, 0},
 { FRAME_END },

 { "VIEW  ", NULL, 11},

 { "Show Red", metro, 23},
```

```
 { "Show Green", metro, 24},
 { "Show Blue", metro, 25},
 { "Show Bit Plane 7", metro, 26},
 { "Show Bit Plane 6", metro, 27},
 { "Show Bit Plane 5", metro, 28},
 { "Show Bit Plane 4", metro, 29},
 { "Show Bit Plane 3", metro, 30},
 { "Show Bit Plane 2", metro, 31},
 { "Show Bit Plane 1", metro, 32},
 { "Show Bit Plane 0", metro, 33},
 { "Display Full Image", metro, 34},
 { FRAME_END },

 { "ANALYSIS  ", NULL, 12},

 { "Calculate Entropy", analysis0, 0},
 { "Chop Image File", analysis5, 0},
 { "Compare Files", analysis1, 0},
 { "Compression Ratio", analysis4, 0},
 { "Difference Images", analysis7,0},
 { "Histogram", analysis3, 0},
 { "Mean Squared Error",analysis2, 0},
 { "Paste Image Files", analysis6, 0},

 { FRAME_END },

 { "LOSSLESS  ", NULL, 13},

 { "Compression", comp1, 0},
 { "Decompression",decomp1, 0},
 { FRAME_END },

 { "LOSSY  ", NULL, 14},

 { "Compression", comp2, 0},
 { "Decompression", decomp2, 0},
 { FRAME_END },

 { "OPTIONS", NULL, 15},

 { "Set Video Mode", metro, 35},
 { "Set Lossy Mode", lmode, 0},
 { "Set Menu Color", mmode, 0},
 { "Make Palette", metro, 36},
 { FRAME_END },
 { FRAME_END }
 };

/*************************************************************************/
/*************************************************************************/
main()
```

```
{
 int i,j,k,l,m,p,ii,jj,kk,mm,red[256],green[256],blue[256];

 FILE *inpalette;

 void doinv(int ixindex, int data1[128][8][8], float c[8][8]);
 void doform(int ixindex, int data1[128][8][8], float temp[8][8], float c[8][8]);
 int  compdecomp(int flag);
 void transform(int flag1);
 void getout(void);
 void rname(void);
 void copyf(void);
 void delecef(void);
 int comp1(void);
 int decomp1(void);
 int comp2(void);
 int decomp2(void);
 int pasteh(int idata);
 int pastev(int idata);
 int analysis0(void);
 int analysis1(void);
 int analysis2(void);
 int analysis3(void);
 int analysis4(void);
 int analysis5(void);
 int analysis6(void);
 int analysis7(void);
 int lmode(int idata);
 int mmode(int idata);
 int metro(char *sdata, int data);
 int file(char *sdata, int data);
 void graphics1(int idata, int mask, int ch);

/*****************************MENU*********************************/
Build_Mode();
vga_code=Which_VGA();
VGA_id = vga_code;
if(vga_code<0)
        {
        SetMode(3);
        strcpy(message_string,"ERROR Exit 100:  Can't Detect VGA");
        Message_On_Screen(message_string);
        exit(1);
        }
graphics = 1;
Number_Color = 256;
for ( i = 1; i<=4; i++)
        {
        modes[i] = 0;
        }
width = 1024;
```

```
height = 768;
mode = Find_Mode(width,height,Number_Color,graphics);
if (mode != -1) modes[4] = mode;
width = 800;
height = 600;
mode = Find_Mode(width,height,Number_Color,graphics);
if (mode != -1) modes[3] = mode;
width = 640;
height = 480;
mode = Find_Mode(width,height,Number_Color,graphics);
if (mode != -1) modes[2] = mode;
width = 320;
height = 200;
mode = Find_Mode(width,height,Number_Color,graphics);
if (modes[4] != 0) mode = modes[4];
else if (modes[3] != 0) mode = modes[3];
     else if (modes[2] != 0) mode = modes[2];
          else mode = modes[1];
if (modes[4] != 0) width = 1024;
else if (modes[3] != 0) width = 800;
     else if (modes[2] != 0) width = 640;
          else width = 320;

inpalette=fopen("graytest.pal","r+b");
p = 0;
while (!feof(inpalette))
        {
        fscanf (inpalette,"%c",&stringpall[p]);
        p = p + 1;
        }
for ( ii = 0; ii<=255; ii++)
        {
        red[ii] = stringpall[ii];
        }
for ( jj = 256; jj<=511; jj++)
        {
        l = jj - 256;
        green[l] = stringpall[jj];
        }
for ( kk = 512; kk<=767; kk++)
        {
        m = kk - 512;
        blue[m] = stringpall[kk];
        }
outp(0x3c8,0);
for ( mm = 0; mm<=255; mm++)
         {
         outp(0x3c9,red[mm]/4);
         outp(0x3c9,green[mm]/4);
         outp(0x3c9,blue[mm]/4);
         }
```

```
Set_Mode(3);

start:
disp_Init(def_ModeCurrent, NULL);
if (colorflag == 0) frame = frame_Open(main_frame, bd_1, 0x04, 0x47, 0x07);
if (colorflag == 1) frame = frame_Open(main_frame, bd_1, 0x02, 0x7A, 0x07);
if (colorflag == 2) frame = frame_Open(main_frame, bd_1, 0x01, 0x7B, 0x07);
if (colorflag == 3) frame = frame_Open(main_frame, bd_1, 0x07, 0x70, 0x07);
frame_Repaint(frame);
frame_Go(frame, ' ', NULL);
frame_Close(frame);
disp_Close();
goto start;
}
```

```
#include <stdio.h>
#include <math.h>

/************************** START DOFORM16 ************************/
void doform16(ixindex, data16, temp16, c16)
int ixindex;
int data16[32][16][16];
float temp16[16][16];
float c16[16][16];
{
int i,l,m,n;
float temp1;

for ( i = 0; i<ixindex; i++)
        {
        for ( l = 0; l<16; l++)
                {
                temp1 = 0;
                for ( m = 0; m<16; m++)
                        {
                        for ( n = 0; n<16; n++)
                                {
                                temp1 += (float) (data16[i][m][n]) * c16[l][n];
                                }
                        temp16[m][l] = temp1;
                        temp1 = 0;
                        }
                }
        for ( l = 0; l<16; l++)
                {
                temp1 = 0;
                for (m = 0; m<16; m++)
                        {
                        for ( n = 0; n<16; n++)
                                {
                                temp1 += c16[m][n] * temp16[n][l];
                                }
                        if (temp1 >= 0)
                                {
                                data16[i][m][l] = (int) (temp1 / 16.0 + 0.5);
                                }
                        else
                                {
                                data16[i][m][l] = (int) (temp1 / 16.0 - 0.5);
                                }
                        temp1 = 0;
                        }
```

```
                }
        }
}
/***************************** END DOFORM16 ***************************/

/***************************** START DOINV16 ***************************/
void doinv16(ixindex, data16, c16)

   int ixindex;
   int data16[32][16][16];
   float c16[16][16];

   {
   int a,b,q,d;
   float temp1, temp[16][16];

    for (a = 0; a<ixindex; a++)
       {
       for (b = 0; b<16; b++)
          {
          temp1 = 0.0;
          for (q = 0; q<16; q++)
             {
             for (d = 0; d<16; d++)
                {
                temp1 += (float) (data16[a][q][d]) * c16[d][b];
                }
             temp[q][b] = temp1;
             temp1 = 0.0;
             }
          }
       for (b = 0; b<16; b++)
          {
          temp1 = 0;
          for (q = 0; q<16; q++)
             {
             for (d = 0; d<16; d++)
                {
                temp1 += c16[d][q] * temp[d][b];
                }
             if (temp1 >= 0)
                {
                data16[a][q][b] = (int) (temp1 * 16 + 0.5);
                }
             else
                {
                data16[a][q][b] = (int) (temp1 * 16 - 0.5);
                }
             temp1 = 0;
             }
        }
```

```
      }
   }


/***************************** END DOINV16 ****************************/

void lossy16(flag1,data11)
int flag1,data11;
{
int ixindex,i,j,k,l,n,mm,x,y,threshold,xkey,ykey,key,key1,
    tempr,data16[32][16][16],xindex,yindex,width3,limit,
    height3,extflag=0,nxindex,zone,interm,width4;
float c16[16][16],temp16[16][16],interm1,interm2;
char string6[80],string7[80],string8[80],string9[80];
unsigned char cp1,cp2,cp3,cp4,cp5,cp6;

FILE *input_file1;
FILE *input_file2;
FILE *input_file3;
FILE *output_file1;
FILE *output_file2;
FILE *out1;
FILE *out2;
FILE *out3;

Set_Mode(3);

if (flag1 == 0) printf("COMPRESSION: ");
if (flag1 == 1) printf("DECOMPRESSION: ");
if (data11 == 15) printf("COSINE-BLACK & WHITE (16 X 16)\n\n");
if (data11 == 18) printf("COSINE-COLOR (16 X 16)\n\n");
if (data11 == 25) printf("HADAMARD-BLACK & WHITE (16 X 16)\n\n");
if (data11 == 28) printf("HADAMARD-COLOR (16 X 16)\n\n");
if (data11 == 35) printf("SINE-BLACK & WHITE (16 X 16)\n\n");
if (data11 == 38) printf("SINE-COLOR (16 X 16)\n\n");

restart20:
printf("Type name of input file: ");
scanf("%s",string6);

if (flag1 == 1)
{
key = 0;
i = 0;
while (string6[i] != '.' && i != 79)
        {
        i = i + 1;
        if (string6[i] == '.') key = i;
        }
string6[key + 1] = '1';
}
```

```
input_file1 = fopen(string6, "r+b");

if (input_file1 == (FILE *) NULL)
        {
        printf("");
        string6[key + 1] = 'd';
        input_file1 = fopen(string6, "r+b");
        }
if (input_file1 == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart20;
        }

printf("\nType name of output file: ");
scanf("%s",string7);

printf("\nEnter width: ");
scanf("%i",&width3);
printf("\nEnter height: ");
scanf("%i",&height3);

if (flag1 == 0)
        {
        printf("\nEnter threshold: ");
        scanf("%i",&threshold);
        restart31:
        printf("\nEnter zonal filter level: ");
        scanf("%i",&zone);
        if (zone > 16 || zone <= 0)
                {
                printf("\nRANGE: 1 - 16!!");
                goto restart31;
                }
        }

if (flag1 == 0)
{
if (data11 == 18 || data11 == 28 || data11 == 38)
        {
        out1 = fopen("Y", "w+b");
        out2 = fopen("Cb","w+b");
        out3 = fopen("Cr","w+b");
        while (!feof(input_file1))
                {
                fscanf (input_file1,"%c",&cp1);
                cp4 = cp1 & 224;
                interm = cp4;
                interm1 = interm;
                interm2 = interm1 * .299;
                cp5 = (cp1 & 28) * 8;
```

```
                interm = cp5;
                interm1 = interm;
                interm2 = interm2 + interm1 * .587;
                cp6 = (cp1 & 3) * 64;
                interm = cp6;
                interm1 = interm;
                interm2 = interm2 + interm1 * .114;
                interm = (interm2 + .5);
                cp6 = interm;
                putc(cp6,out1);
                cp4 = cp1 & 224;
                interm = cp4;
                interm1 = interm;
                interm2 = interm1 * -.16874;
                cp5 = (cp1 & 28) * 8;
                interm = cp5;
                interm1 = interm;
                interm2 = interm2 + interm1 * -.33126;
                cp6 = (cp1 & 3) * 64;
                interm = cp6;
                interm1 = interm;
                interm2 = interm2 + interm1 * .5;
                if (interm2 < 0) interm = (interm2 +.5);
                else interm = (interm2 - .5);
                cp6 = interm + 128;
                putc(cp6,out2);
                cp4 = cp1 & 224;
                interm = cp4;
                interm1 = interm;
                interm2 = interm1 * .5;
                cp5 = (cp1 & 28) * 8;
                interm = cp5;
                interm1 = interm;
                interm2 = interm2 + interm1 * -.41869;
                cp6 = (cp1 & 3) * 64;
                interm = cp6;
                interm1 = interm;
                interm2 = interm2 + interm1 * -.08131;
                if (interm2 > 0) interm = (interm2 +.5);
                else interm = (interm2 - .5);
                cp6 = interm + 128;
                putc(cp6,out3);
                }
        fclose(input_file1);
        fclose(out1);
        fclose(out2);
        fclose(out3);
        }
}

output_file1 = fopen(string7, "w+b");
```

```
for (x = 0; x < 80; x++)
        {
        string8[x] = string7[x];
        }
if (flag1 == 0)
{
key = 0;
i = 0;
while (string8[i] != '.' && i != 79)
        {
        i = i + 1;
        if (string8[i] == '.') key = i;
        }
string8[key + 1] = '1';
output_file2 = fopen(string8, "w+b");
}

if (data11 == 18 || data11 == 28 || data11 == 38)
        {
        fclose(output_file1);
        fclose(output_file2);
        }

if (width3 > 512) extflag = 1;

xindex = width3/16;
yindex = height3/16;

if (data11 == 15 || data11 == 18) goto cosinetran16;
if (data11 == 25 || data11 == 28) goto hadamard16;
if (data11 == 35 || data11 == 38) goto sinetran16;

cosinetran16:

for ( k = 0; k<=15; k++)
        {
        for ( n = 0; n<=15; n++)
                {
                if (k == 0) c16[k][n] = 1.0 / 4.0;
                else c16[k][n] = 0.3535534 * cos(3.14159 * (2 * n + 1) * k / 32.0);
                }
        }

goto continue16;

hadamard16:
c16[0][0] = 1; c16[0][1] = 1; c16[0][2] = 1; c16[0][3] = 1;
c16[0][4] = 1; c16[0][5] = 1; c16[0][6] = 1; c16[0][7] = 1;
c16[1][0] = 1; c16[1][1] = -1; c16[1][2] = 1; c16[1][3] = -1;
c16[1][4] = 1; c16[1][5] = -1; c16[1][6] = 1; c16[1][7] = -1;
c16[2][0] = 1; c16[2][1] = 1; c16[2][2] = -1; c16[2][3] = -1;
```

```
c16[2][4] = 1; c16[2][5] = 1; c16[2][6] = -1; c16[2][7] = -1;
c16[3][0] = 1; c16[3][1] = -1; c16[3][2] = -1; c16[3][3] = 1;
c16[3][4] = 1; c16[3][5] = -1; c16[3][6] = -1; c16[3][7] = 1;
c16[4][0] = 1; c16[4][1] = 1; c16[4][2] = 1; c16[4][3] = 1;
c16[4][4] = -1; c16[4][5] = -1; c16[4][6] = -1; c16[4][7] = -1;
c16[5][0] = 1; c16[5][1] = -1; c16[5][2] = 1; c16[5][3] = -1;
c16[5][4] = -1; c16[5][5] = 1; c16[5][6] = -1; c16[5][7] = 1;
c16[6][0] = 1; c16[6][1] = 1; c16[6][2] = -1; c16[6][3] = -1;
c16[6][4] = -1; c16[6][5] = -1; c16[6][6] = 1; c16[6][7] = 1;
c16[7][0] = 1; c16[7][1] = -1; c16[7][2] = -1; c16[7][3] = 1;
c16[7][4] = -1; c16[7][5] = 1; c16[7][6] = 1; c16[7][7] = -1;
for ( j = 0; j<=7; j++)
        {
        for ( i = 0; i<=7; i++)
                {
                c16[i+8][j] = c16[i][j];
                c16[i][j+8] = c16[i][j];
                c16[i+8][j+8] = -c16[i][j];
                }
        }
for ( i = 0; i<=15; i++)
        {
        for ( j = 0; j<=15; j++)
                {
                c16[i][j] = c16[i][j] / 4.0;
                }
        }
goto continue16;

sinetran16:
for (k = 0; k<=15; k++)
        {
        for (n = 0; n<=15; n++)
                {
                c16[k][n] = .342997 * sin(3.14159 * (n + 1) * (k + 1)/17.0);
                }
        }

continue16:
for (mm = 0; mm < 80; mm++)
        {
        string9[mm] = string7[mm];
        }
if (data11 == 18 || data11 == 28 || data11 == 38) limit = 3;
else limit = 1;
for (l = 0; l < limit; l++)
{
if (flag1 == 0)
        {
        if (data11 == 18 || data11 == 28 || data11 == 38)
                {
```

```
if (l == 0)
        {
        for (mm = 0; mm < 80; mm++)
                {
                string6[mm] = 0;
                }
        string6[0] = 'Y';
        input_file1 = fopen("Y", "r+b");
        key = 0;
        i = 0;
        while (string7[i] != '.' && i != 79)
                {
                i = i + 1;
                if (string7[i] == '.') key = i;
                }
        string7[key + 1] = 'y';
        string7[key + 2] = 'y';
        string7[key + 3] = 'y';
        output_file1 = fopen(string7, "w+b");
        for (mm = 0; mm < 80; mm++)
                {
                string8[mm] = string7[mm];
                }
        string8[key + 1] = '1';
        output_file2 = fopen(string8, "w+b");
        }

if (l == 1)
        {
        for (mm = 0; mm < 80; mm++)
                {
                string6[mm] = 0;
                }
        string6[0] = 'C';
        string6[1] = 'b';
        input_file1 = fopen("Cb", "r+b");
        key = 0;
        i = 0;
        while (string7[i] != '.' && i != 79)
                {
                i = i + 1;
                if (string7[i] == '.') key = i;
                }
        string7[key + 1] = 'c';
        string7[key + 2] = 'c';
        string7[key + 3] = 'b';
        output_file1 = fopen(string7, "w+b");
        for (mm = 0; mm < 80; mm++)
                {
                string8[mm] = string7[mm];
                }
```

```
                        string8[key + 1] = '1';
                        output_file2 = fopen(string8, "w+b");
                        }

                if (l == 2)
                        {
                        for (mm = 0; mm < 80; mm++)
                                {
                                string6[mm] = 0;
                                }
                        string6[0] = 'C';
                        string6[1] = 'r';
                        input_file1 = fopen("Cr", "r+b");
                        key = 0;
                        i = 0;
                        while (string7[i] != '.' && i != 79)
                                {
                                i = i + 1;
                                if (string7[i] == '.') key = i;
                                }
                        string7[key + 1] = 'c';
                        string7[key + 2] = 'c';
                        string7[key + 3] = 'r';
                        output_file1 = fopen(string7, "w+b");
                        for (mm = 0; mm < 80; mm++)
                                {
                                string8[mm] = string7[mm];
                                }
                        string8[key + 1] = '1';
                        output_file2 = fopen(string8, "w+b");
                        }
                }
        }
if (flag1 == 1)
        {
        if (data11 == 18 || data11 == 28 || data11 == 38)
                {
                if (l == 0)
                        {
                        key = 0;
                        i = 0;
                        while (string6[i] != '.' && i != 79)
                                {
                                i = i + 1;
                                if (string6[i] == '.') key = i;
                                }
                        string6[key + 1] = '1';
                        string6[key + 2] = 'y';
                        string6[key + 3] = 'y';
                        input_file1 = fopen(string6, "r+b");
                        for (mm = 0; mm < 80; mm++)
```

```
                {
                string7[mm] = 0;
                }
        string7[0] = 'Y';
        string7[1] = '1';
        output_file1 = fopen("Y1", "w+b");
        }

if (l == 1)
        {
        key = 0;
        i = 0;
        while (string6[i] != '.' && i != 79)
                {
                i = i + 1;
                if (string6[i] == '.') key = i;
                }
        string6[key + 1] = '1';
        string6[key + 2] = 'c';
        string6[key + 3] = 'b';
        input_file1 = fopen(string6, "r+b");
        for (mm = 0; mm < 80; mm++)
                {
                string7[mm] = 0;
                }
        string7[0] = 'C';
        string7[1] = 'b';
        string7[2] = '1';
        output_file1 = fopen("Cb1", "w+b");
        }

if (l == 2)
        {
        key = 0;
        i = 0;
        while (string6[i] != '.' && i != 79)
                {
                i = i + 1;
                if (string6[i] == '.') key = i;
                }
        string6[key + 1] = '1';
        string6[key + 2] = 'c';
        string6[key + 3] = 'r';
        input_file1 = fopen(string6, "r+b");
        for (mm = 0; mm < 80; mm++)
                {
                string7[mm] = 0;
                }
        string7[0] = 'C';
        string7[1] = 'r';
        string7[2] = '1';
```

```
                        output_file1 = fopen("Cr1", "w+b");
                        }
                }
        }

if (extflag == 0) goto next;
width4 = xindex*16;
if (width4 != width3) width4 = width4 + 16;
printf("\n.");
if (16 * yindex != height3) yindex = yindex + 1;
for( j = 0; j<=yindex-1; j++)
        {
        if (flag1 == 0)
        {
        printf(".");
        for ( y = 0; y<=15; y++)
                {
                for ( i = 0; i<=31; i++)
                        {
                        for ( x = 0; x<=15; x++)
                                {
                                if (j * 16 + (y + 1) < height3)
                                        {
                                        data16[i][y][x] = 0;
                                        }
                                else
                                        {
                                        ykey = y;
                                        fscanf(input_file1,"%c",&cp1);
                                        data16[i][y][x] = cp1 - 128;
                                        }
                                }
                        }
                        x = 512;
                        while (x != width3)
                                {
                                fscanf(input_file1,"%c",&cp1);
                                x=x+1;
                                }
                }
        }
        if (flag1 == 1)
        {
        printf(".");
        for ( y = 0; y<=15; y++)
                {
                if (j * 16 + (y + 1) <= height3) ykey = y;
                for ( i = 0; i<=31; i++)
                        {
                        for ( x = 0; x<=15; x++)
                                {
```

```
                        fscanf(input_file1,"%c",&cp1);
                        data16[i][y][x] = cp1 - 128;
                        }
                }
                x = 512;
                while (x != width4)
                        {
                        fscanf(input_file1,"%c",&cp1);
                        x=x+1;
                        }
        }
}

if (flag1 == 0) doform16(32, data16, temp16, c16);
if (flag1 == 1) doinv16(32, data16, c16);

if (flag1 == 0)
{
for ( y = 0; y<=15; y++)
        {
        for ( i = 0; i<=31; i++)
                {
                for ( x = 0; x<=15; x++)
                        {
                        if (data16[i][y][x] >= -threshold && data16[i][y][x]
<= threshold) data16[i][y][x] = 0;
                        if (data16[i][y][x] > 127) data16[i][y][x] = 127;
                        if (data16[i][y][x] < -128) data16[i][y][x] = -128;
                        cp1 = data16[i][y][x] + 128;
                        if (j * 16 + (y + 1) <= height3) putc (cp1,output_file1);
                        putc (cp1,output_file2);
                        }
                }
        }
}
if (flag1 == 1)
{
for ( y = 0; y<=ykey; y++)
        {
        for ( i = 0; i<=31; i++)
                {
                for ( x = 0; x<=15; x++)
                        {
                        if (data16[i][y][x] > 127) data16[i][y][x] = 127;
                        if (data16[i][y][x] < -128) data16[i][y][x] = -128;
                        cp1 = data16[i][y][x] + 128;
                        putc (cp1,output_file1);
                        }
                }
        }
}
```

```
        }
fclose(input_file1);
fclose(output_file1);
if (flag1 == 0) fclose(output_file2);

xindex = width3/16;
yindex = height3/16;
input_file1 = fopen(string6, "r+b");
output_file1 = fopen("temp1", "w+b");
if (flag1 == 0) output_file2 = fopen("temp2", "w+b");
nxindex = xindex - 32;


next:
if (extflag == 0) nxindex = xindex;
printf("\n.");
if (16 * yindex != height3) yindex = yindex + 1;
for( j = 0; j<=yindex-1; j++)
        {
        if (flag1 == 0)
        {
        printf(".");
        for ( y = 0; y<=15; y++)
                {
                if (extflag == 1)
                        {
                        for (k = 0; k < 512; k++) fscanf(input_file1,"%c",&cp1);
                        }
                for ( i = 0; i<=nxindex-1; i++)
                        {
                        for ( x = 0; x<=15; x++)
                                {
                                if (j * 16 + (y + 1) > height3)
                                        {
                                        data16[i][y][x] = 0;
                                        }
                                else
                                        {
                                        ykey = y;
                                        fscanf(input_file1,"%c",&cp1);
                                        data16[i][y][x] = cp1 - 128;
                                        }
                                }
                        }
                        if ( 16 * xindex != width3)
                                {
                                xkey = 16 * xindex;
                                i = nxindex;
                                x = 0;
                                while (xkey != width3)
                                        {
```

```
                                if (j * 16 + (y + 1) > height3)
                                        {
                                        data16[i][y][x] = 0;
                                        }
                                else
                                        {
                                        fscanf(input_file1,"%c",&cp1);
                                        data16[i][y][x] = cp1 - 128;
                                        }
                                x = x + 1;
                                xkey = xkey + 1;
                                }
                        while (x != 16)
                                {
                                data16[i][y][x] = 0;
                                x = x + 1;
                                }
                        }
        }
}
if (flag1 == 1)
{
printf(".");
for ( y = 0; y<=15; y++)
        {
        if (j * 16 + (y + 1) <= height3) ykey = y;
        if (extflag == 1)
                {
                for (k = 0; k < 512; k++) fscanf(input_file1,"%c",&cp1);
                }
        for ( i = 0; i<=nxindex-1; i++)
                {
                for ( x = 0; x<=15; x++)
                        {
                        fscanf(input_file1,"%c",&cp1);
                        data16[i][y][x] = cp1 - 128;
                        }
                }
                if ( 16 * xindex != width3)
                        {
                        xkey = 16 * xindex;
                        i = nxindex;
                        x = 0;
                        while (xkey != width3)
                                {
                                fscanf(input_file1,"%c",&cp1);
                                data16[i][y][x] = cp1 - 128;
                                x = x + 1;
                                xkey = xkey + 1;
                                }
                        while (x != 16)
```

```
                                        {
                                        fscanf(input_file1,"%c",&cp1);
                                        data16[i][y][x] = cp1 - 128;
                                        x = x + 1;
                                        }
                                }
                        }
                }

                ixindex = nxindex;
                if (16 * xindex != width3) ixindex = ixindex + 1;
                if (flag1 == 0) doform16(ixindex, data16, temp16, c16);
                if (flag1 == 1) doinv16(ixindex, data16, c16);

                if (flag1 == 0)
                {
                for ( y = 0; y<=15; y++)
                        {
                        for ( i = 0; i<=nxindex-1; i++)
                                {
                                for ( x = 0; x<=15; x++)
                                        {
                                        if (data16[i][y][x] >= -threshold && data16[i][y][x]
<= threshold) data16[i][y][x] = 0;
                                        if (data16[i][y][x] > 127) data16[i][y][x] = 127;
                                        if (data16[i][y][x] < -128) data16[i][y][x] = -128;
                                        cp1 = data16[i][y][x] + 128;
                                        if (j * 16 + (y + 1) <= height3) putc (cp1,output_file1);
                                        putc (cp1,output_file2);
                                        }
                                }
                                if ( 16 * xindex != width3)
                                        {
                                        xkey = 16 * xindex;
                                        i = nxindex;
                                        x = 0;
                                        while (xkey != width3)
                                                {
                                                if (data16[i][y][x] >= -threshold && data16[i][y][x]
<= threshold) data16[i][y][x] = 0;
                                                if (data16[i][y][x] > 127) data16[i][y][x] = 127;
                                                if (data16[i][y][x] < -128) data16[i][y][x] = -128;
                                                cp1 = data16[i][y][x] + 128;
                                                if (j * 16 + (y + 1) <= height3) putc (cp1,output_file1);
                                                putc (cp1,output_file2);
                                                x = x + 1;
                                                xkey = xkey + 1;
                                                }
                                        while (x != 16)
                                                {
```

```
                                        if (data16[i][y][x] >= -threshold && data16[i][y][x]
<= threshold) data16[i][y][x] = 0;
                                        if (data16[i][y][x] > 127) data16[i][y][x] = 127;
                                        if (data16[i][y][x] < -128) data16[i][y][x] = -128;
                                        cp1 = data16[i][y][x] + 128;
                                        putc (cp1,output_file2);
                                        x = x + 1;
                                        }
                                    }
                        }
                }
                if (flag1 == 1)
                {
                for ( y = 0; y<=ykey; y++)
                        {
                        for ( i = 0; i<=nxindex-1; i++)
                                {
                                for ( x = 0; x<=15; x++)
                                        {
                                        if (data16[i][y][x] > 127) data16[i][y][x] = 127;
                                        if (data16[i][y][x] < -128) data16[i][y][x] = -128;
                                        cp1 = data16[i][y][x] + 128;
                                        putc (cp1,output_file1);
                                        }
                                }
                                if ( 16 * xindex != width3)
                                        {
                                        xkey = 16 * xindex;
                                        i = nxindex;
                                        x = 0;
                                        while (xkey != width3)
                                                {
                                                if (data16[i][y][x] > 127) data16[i][y][x] = 127;
                                                if (data16[i][y][x] < -128) data16[i][y][x] = -128;
                                                cp1 = data16[i][y][x] + 128;
                                                putc (cp1,output_file1);
                                                x = x + 1;
                                                xkey = xkey + 1;
                                                }
                                        }
                        }
                }
                }
fclose(input_file1);
fclose(output_file1);
if (flag1 == 0) fclose(output_file2);

if (extflag == 0 && zone != 16)
{
if (flag1 == 0)
{
```

```
input_file1 = fopen(string8, "r+b");
output_file1 = fopen("temp1", "w+b");

if (xindex*16 != width3) xindex = xindex + 1;

for (y = 0; y < yindex; y++)
        {
        for (i = 0; i < 16; i++)
        {
        for (x = 0; x < xindex; x++)
                {
                for (j = 0; j < 16; j++)
                {
                fscanf(input_file1,"%c",&cp1);
                putc (cp1,output_file1);
                }
                }
        }
        }

fclose(input_file1);
fclose(output_file1);

input_file1 = fopen("temp1", "r+b");
output_file1 = fopen(string8, "w+b");

for (y = 0; y < yindex; y++)
        {
        for (i = 0; i < 16; i++)
        {
        for (x = 0; x < xindex; x++)
                {
                for (j = 0; j < 16; j++)
                {
                fscanf(input_file1,"%c",&cp1);
                if (i >= zone || j >= zone) cp1 = 128;
                putc (cp1,output_file1);
                }
                }
        }
        }

fclose(input_file1);
fclose(output_file1);
}
}


if (extflag == 1)
{
input_file1 = fopen(string7, "r+b");
```

```
input_file2 = fopen("temp1", "r+b");
output_file1 = fopen("temp3", "w+b");

for (y = 0; y < height3; y++)
        {
        for (x = 0; x < width3; x++)
                {
                if (x < 512) fscanf(input_file1,"%c",&cp1);
                else fscanf(input_file2,"%c",&cp1);
                putc (cp1,output_file1);
                }
        }
fclose(input_file1);
fclose(input_file2);
fclose(output_file1);

input_file1 = fopen("temp3", "r+b");
output_file1 = fopen(string7, "w+b");

for (y = 0; y < height3; y++)
        {
        for (x = 0; x < width3; x++)
                {
                fscanf(input_file1,"%c",&cp1);
                putc (cp1,output_file1);
                }
        }

fclose(input_file1);
fclose(output_file1);

if (flag1 == 0)
{
input_file1 = fopen(string8, "r+b");
input_file2 = fopen("temp2", "r+b");
output_file1 = fopen("temp3", "w+b");


if (xindex*16 != width3) xindex = xindex + 1;

for (y = 0; y < yindex*16; y++)
        {
        for (x = 0; x < xindex*16; x++)
                {
                if (x < 512) fscanf(input_file1,"%c",&cp1);
                else fscanf(input_file2,"%c",&cp1);
                putc (cp1,output_file1);
                }
        }
fclose(input_file1);
fclose(input_file2);
```

```
fclose(output_file1);

input_file1 = fopen("temp3", "r+b");
output_file1 = fopen(string8, "w+b");

for (y = 0; y < yindex; y++)
        {
        for (i = 0; i < 16; i++)
        {
        for (x = 0; x < xindex; x++)
                {
                for (j = 0; j < 16; j++)
                {
                fscanf(input_file1,"%c",&cp1);
                if (i >= zone || j >= zone) cp1 = 128;
                putc (cp1,output_file1);
                }
                }
        }
        }

fclose(input_file1);
fclose(output_file1);
}

remove("temp1");
remove("temp2");
remove("temp3");
}
}

if (data11 == 18 || data11 == 28 || data11 == 38)
        {
        if (flag1 == 1)
        {
        input_file1 = fopen("Y1", "r+b");
        input_file2 = fopen("Cb1", "r+b");
        input_file3 = fopen("Cr1", "r+b");
        output_file1 = fopen(string9, "w+b");
        while (!feof(input_file1) && !feof(input_file2) && !feof(input_file3))
                {
                fscanf (input_file1,"%c",&cp1);
                fscanf (input_file2,"%c",&cp2);
                fscanf (input_file3,"%c",&cp3);
                interm = cp1;
                interm2 = interm;
                interm = cp3 - 128;
                interm1 = interm;
                interm2 = interm2 + interm1 * 1.402;
                interm = (interm2 + .5);
                if (interm < 0) interm = 0;
```

```
        cp4 = interm;
        if (cp4 > 208) cp4 = 224;
        else if (cp4 > 176) cp4 = 192;
                else if (cp4 > 144) cp4 = 160;
                        else if (cp4 > 112) cp4 = 128;
                                else if (cp4 > 80) cp4 = 96;
                                        else if (cp4 > 48) cp4 = 64;
                                                else if (cp4 > 16) cp4 = 32;
                                                        else cp4 = 0;
        interm = cp1;
        interm2 = interm;
        interm = cp2 - 128;
        interm1 = interm;
        interm1 = interm1 * .34414;
        interm2 = interm2 - interm1;
        interm = cp3 - 128;
        interm1 = interm;
        interm1 = interm1 * .71414;
        interm2 = interm2 - interm1;
        interm = (interm2 + .5);
        if (interm < 0) interm = 0;
        cp5 = interm;
        if (cp5 > 208) cp5 = 224;
        else if (cp5 > 176) cp5 = 192;
                else if (cp5 > 144) cp5 = 160;
                        else if (cp5 > 112) cp5 = 128;
                                else if (cp5 > 80) cp5 = 96;
                                        else if (cp5 > 48) cp5 = 64;
                                                else if (cp5 > 16) cp5 = 32;
                                                        else cp5 = 0;
        cp5 = cp5 / 8;
        interm = cp1;
        interm2 = interm;
        interm = cp2 - 128;
        interm1 = interm;
        interm2 = interm2 + interm1 * 1.772;
        interm = (interm2 + .5);
        if (interm < 0) interm = 0;
        cp6 = interm,
        if (cp6 > 160) cp6 = 192;
        else if (cp6 > 96) cp6 = 128;
                else if (cp6 > 32) cp6 = 64;
                        else cp6 = 0;
        cp6 = cp6 / 64;
        cp1 = cp4 + cp5 + cp6;
        putc (cp1,output_file1);
        }
fclose(input_file1);
fclose(input_file2);
fclose(input_file3);
fclose(output_file1);
```

```
        remove("Y1");
        remove("Cb1");
        remove("Cr1");
        }
        if (flag1 == 0)
        {
        remove("Y");
        remove("Cb");
        remove("Cr");
        }
        }

end1:
printf("");
}
```

```
#include <stdio.h>
#include <math.h>

/************************** START DOFORM32 ************************/
void doform32(ixindex, data32, c32)
int ixindex;
int data32[5][32][32];
float c32[32][32];
{
int i,l,m,n;
float temp1,temp32[32][32];

for ( i = 0; i<ixindex; i++)
        {
        for ( l = 0; l<32; l++)
                {
                temp1 = 0;
                for ( m = 0; m<32; m++)
                        {
                        for ( n = 0; n<32; n++)
                                {
                                temp1 += (float) (data32[i][m][n]) * c32[l][n];
                                }
                        temp32[m][l] = temp1;
                        temp1 = 0;
                        }
                }
        for ( l = 0; l<32; l++)
                {
                temp1 = 0;
                for (m = 0; m<32; m++)
                        {
                        for ( n = 0; n<32; n++)
                                {
                                temp1 += c32[m][n] * temp32[n][l];
                                }
                        if (temp1 >= 0)
                                {
                                data32[i][m][l] = (int) (temp1 / 32.0 + 0.5);
                                }
                        else
                                {
                                data32[i][m][l] = (int) (temp1 / 32.0 - 0.5);
                                }
                        temp1 = 0;
                        }
                }
```

```
        }
}
/***************************** END DOFORM32 ***************************/

/***************************** START DOINV32 **************************/
void doinv32(ixindex, data32, c32)

   int ixindex;
   int data32[5][32][32];
   float c32[32][32];

   {
   int a,b,q,d;
   float temp1, temp[32][32];

    for (a = 0; a<ixindex; a++)
       {
       for (b = 0; b<32; b++)
          {
          temp1 = 0.0;
          for (q = 0; q<32; q++)
             {
             for (d = 0; d<32; d++)
                {
                temp1 += (float) (data32[a][q][d]) * c32[d][b];
                }
             temp[q][b] = temp1;
             temp1 = 0.0;
             }
          }
       for (b = 0; b<32; b++)
          {
          temp1 = 0;
          for (q = 0; q<32; q++)
             {
             for (d = 0; d<32; d++)
                {
                temp1 += c32[d][q] * temp[d][b];
                }
             if (temp1 >= 0)
                {
                data32[a][q][b] = (int) (temp1 * 32 + 0.5);
                }
             else
                {
                data32[a][q][b] = (int) (temp1 * 32 - 0.5);
                }
             temp1 = 0;
             }
          }
       }
```

```
    }


/****************************** END DOINV32 ****************************/

void lossy32(flag1,data11)
int flag1,data11;
{
int ixindex,i,j,k,l,n,mm,x,y,threshold,xkey,ykey,key,key1,
    marker,tempr,xindex,yindex,width3,height3,data32[5][32][32],
    limit,extflag=0,nxindex,zone,interm,width4;
float c32[32][32],temp32[32][32],interm1,interm2;
char string6[80],string7[80],string8[80],string9[80];
unsigned char cp1,cp2,cp3,cp4,cp5,cp6;

FILE *input_file1;
FILE *input_file2;
FILE *input_file3;
FILE *input_file4;
FILE *output_file1;
FILE *output_file2;
FILE *out1;
FILE *out2;
FILE *out3;

Set_Mode(3);

if (flag1 == 0) printf("COMPRESSION: ");
if (flag1 == 1) printf("DECOMPRESSION: ");
if (data11 == 15) printf("COSINE-BLACK & WHITE (32 X 32)\n\n");
if (data11 == 18) printf("COSINE-COLOR (32 X 32)\n\n");
if (data11 == 25) printf("HADAMARD-BLACK & WHITE (32 X 32)\n\n");
if (data11 == 28) printf("HADAMARD-COLOR (32 X 32)\n\n");
if (data11 == 35) printf("SINE-BLACK & WHITE (32 X 32)\n\n");
if (data11 == 38) printf("SINE-COLOR (32 X 32)\n\n");

restart20:
printf("Type name of input file: ");
scanf("%s",string6);

if (flag1 == 1)
{
key = 0;
i = 0;
while (string6[i] != '.' && i != 79)
        {
        i = i + 1;
        if (string6[i] == '.') key = i;
        }
string6[key + 1] = '1';
}
```

```
input_file1 = fopen(string6, "r+b");

if (input_file1 == (FILE *) NULL)
        {
        printf("");
        string6[key + 1] = 'd';
        input_file1 = fopen(string6, "r+b");
        }
if (input_file1 == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart20;
        }

printf("\nType name of output file: ");
scanf("%s",string7);

printf("\nEnter width: ");
scanf("%i",&width3);
printf("\nEnter height: ");
scanf("%i",&height3);

if (width3 > 640)
        {
        printf("\nMAXIMUM ALLOWABLE WIDTH IS 640.");
        getch();
        goto end1;
        }

if (flag1 == 0)
        {
        printf("\nEnter threshold: ");
        scanf("%i",&threshold);
        restart30:
        printf("\nEnter zonal filter level: ");
        scanf("%i",&zone);
        if (zone > 32 || zone <= 0)
                {
                printf("\nRANGE:  1 - 32!!");
                goto restart30;
                }


        }

if (flag1 == 0)
{
if (data11 == 18 || data11 == 28 || data11 == 38)
        {
        out1 = fopen("Y", "w+b");
        out2 = fopen("Cb","w+b");
```

```
out3 = fopen("Cr","w+b");
while (!feof(input_file1))
        {
        fscanf (input_file1,"%c",&cp1);
        cp4 = cp1 & 224;
        interm = cp4;
        interm1 = interm;
        interm2 = interm1 * .299;
        cp5 = (cp1 & 28) * 8;
        interm = cp5;
        interm1 = interm;
        interm2 = interm2 + interm1 * .587;
        cp6 = (cp1 & 3) * 64;
        interm = cp6;
        interm1 = interm;
        interm2 = interm2 + interm1 * .114;
        interm = (interm2 + .5);
        cp6 = interm;
        putc(cp6,out1);
        cp4 = cp1 & 224;
        interm = cp4;
        interm1 = interm;
        interm2 = interm1 * -.16874;
        cp5 = (cp1 & 28) * 8;
        interm = cp5;
        interm1 = interm;
        interm2 = interm2 + interm1 * -.33126;
        cp6 = (cp1 & 3) * 64;
        interm = cp6;
        interm1 = interm;
        interm2 = interm2 + interm1 * .5;
        if (interm2 > 0) interm = (interm2 +.5);
        else interm = (interm2 - .5);
        cp6 = interm + 128;
        putc(cp6,out2);
        cp4 = cp1 & 224;
        interm = cp4;
        interm1 = interm;
        interm2 = interm1 * .5;
        cp5 = (cp1 & 28) * 8;
        interm = cp5;
        interm1 = interm;
        interm2 = interm2 + interm1 * -.41869;
        cp6 = (cp1 & 3) * 64;
        interm = cp6;
        interm1 = interm;
        interm2 = interm2 + interm1 * -.08131;
        if (interm2 > 0) interm = (interm2 +.5);
        else interm = (interm2 - .5);
        cp6 = interm + 128;
        putc(cp6,out3);
```

```
                }
        fclose(input_file1);
        fclose(out1);
        fclose(out2);
        fclose(out3);
        }


output_file1 = fopen(string7, "w+b");
for (x= 0; x < 80; x++)
        {
        string8[x] = string7[x];
        }
if (flag1 == 0)
{
key = 0;
i = 0;
while (string8[i] != '.' && i != 79)
        {
        i = i + 1;
        if (string8[i] == '.') key = i;
        }
string8[key + 1] = '1';
output_file2 = fopen(string8, "w+b");
}

if (data11 == 18 || data11 == 28 || data11 == 38)
        {
        fclose(output_file1);
        fclose(output_file2);
        }

if (width3 > 160) extflag = 1;
if (width3 > 320) extflag = 2;
if (width3 > 480) extflag = 3;

xindex = width3/32;
yindex = height3/32;

if (data11 == 15 || data11 == 18) goto cosinetran32;
if (data11 == 25 || data11 == 28) goto hadamard32;
if (data11 == 35 || data11 == 38) goto sinetran32;

cosinetran32:
for ( k = 0; k<=31; k++)
        {
        for ( n = 0; n<=31; n++)
                {
                if (k == 0) c32[k][n] = 0.1767767;
                else c32[k][n] = 0.25 * cos(3.14159 * (2 * n + 1) * k / 64.0);
                }
```

```
        }

goto continue32;

hadamard32:
c32[0][0] = 1; c32[0][1] = 1; c32[0][2] = 1; c32[0][3] = 1;
c32[0][4] = 1; c32[0][5] = 1; c32[0][6] = 1; c32[0][7] = 1;
c32[1][0] = 1; c32[1][1] = -1; c32[1][2] = 1; c32[1][3] = -1;
c32[1][4] = 1; c32[1][5] = -1; c32[1][6] = 1; c32[1][7] = -1;
c32[2][0] = 1; c32[2][1] = 1; c32[2][2] = -1; c32[2][3] = -1;
c32[2][4] = 1; c32[2][5] = 1; c32[2][6] = -1; c32[2][7] = -1;
c32[3][0] = 1; c32[3][1] = -1; c32[3][2] = -1; c32[3][3] = 1;
c32[3][4] = 1; c32[3][5] = -1; c32[3][6] = -1; c32[3][7] = 1;
c32[4][0] = 1; c32[4][1] = 1; c32[4][2] = 1; c32[4][3] = 1;
c32[4][4] = -1; c32[4][5] = -1; c32[4][6] = -1; c32[4][7] = -1;
c32[5][0] = 1; c32[5][1] = -1; c32[5][2] = 1; c32[5][3] = -1;
c32[5][4] = -1; c32[5][5] = 1; c32[5][6] = -1; c32[5][7] = 1;
c32[6][0] = 1; c32[6][1] = 1; c32[6][2] = -1; c32[6][3] = -1;
c32[6][4] = -1; c32[6][5] = -1; c32[6][6] = 1; c32[6][7] = 1;
c32[7][0] = 1; c32[7][1] = -1; c32[7][2] = -1; c32[7][3] = 1;
c32[7][4] = -1; c32[7][5] = 1; c32[7][6] = 1; c32[7][7] = -1;

for ( j = 0; j<=7; j++)
        {
        for ( i = 0; i<=7; i++)
                {
                c32[i+8][j] = c32[i][j];
                c32[i][j+8] = c32[i][j];
                c32[i+8][j+8] = -c32[i][j];
                }
        }

for ( j = 0; j<=15; j++)
        {
        for ( i = 0; i<=15; i++)
                {
                c32[i+16][j] = c32[i][j];
                c32[i][j+16] = c32[i][j];
                c32[i+16][j+16] = -c32[i][j];
                }
        }


for ( i = 0; i<=31; i++)
        {
        for ( j = 0; j<=31; j++)
                {
                c32[i][j] = c32[i][j] / 5.656854;
                }
        }
goto continue32;
```

```
sinetran32:
for (k = 0; k<=31; k++)
        {
        for (n = 0; n<=31; n++)
                {
                c32[k][n] = 0.246183 * sin(3.14159 * (n + 1) * (k + 1)/33.0);
                }
        }

continue32:
for (mm = 0; mm < 80; mm++)
        {
        string9[mm] = string7[mm];
        }
if (data11 == 18 || data11 == 28 || data11 == 38) limit = 3;
else limit = 1;
for (l = 0; l < limit; l++)
{
if (flag1 == 0)
        {
        if (data11 == 18 || data11 == 28 || data11 == 38)
                {
                if (l == 0)
                        {
                        for (mm = 0; mm < 80; mm++)
                                {
                                string6[mm] = 0;
                                }
                        string6[0] = 'Y';
                        input_file1 = fopen("Y", "r+b");
                        key = 0;
                        i = 0;
                        while (string7[i] != '.' && i != 79)
                                {
                                i = i + 1;
                                if (string7[i] == '.') key = i;
                                }
                        string7[key + 1] = 'y';
                        string7[key + 2] = 'y';
                        string7[key + 3] = 'y';
                        output_file1 = fopen(string7, "w+b");
                        for (mm = 0; mm < 80; mm++)
                                {
                                string8[mm] = string7[mm];
                                }
                        string8[key + 1] = '1';
                        output_file2 = fopen(string8, "w+b");
                        }

                if (l == 1)
```

```
        {
        for (mm = 0; mm < 80; mm++)
                {
                string6[mm] = 0;
                }
        string6[0] = 'C';
        string6[1] = 'b';
        input_file1 = fopen("Cb", "r+b");
        key = 0;
        i = 0;
        while (string7[i] != '.' && i != 79)
                {
                i = i + 1;
                if (string7[i] == '.') key = i;
                }
        string7[key + 1] = 'c';
        string7[key + 2] = 'c';
        string7[key + 3] = 'b';
        output_file1 = fopen(string7, "w+b");
        for (mm = 0; mm < 80; mm++)
                {
                string8[mm] = string7[mm];
                }
        string8[key + 1] = '1';
        output_file2 = fopen(string8, "w+b");
        }

if (l == 2)
        {
        for (mm = 0; mm < 80; mm++)
                {
                string6[mm] = 0;
                }
        string6[0] = 'C';
        string6[1] = 'r';
        input_file1 = fopen("Cr", "r+b");
        key = 0;
        i = 0;
        while (string7[i] != '.' && i != 79)
                {
                i = i + 1;
                if (string7[i] == '.') key = i;
                }
        string7[key + 1] = 'c';
        string7[key + 2] = 'c';
        string7[key + 3] = 'r';
        output_file1 = fopen(string7, "w+b");
        for (mm = 0; mm < 80; mm++)
                {
                string8[mm] = string7[mm];
                }
```

```
                        string8[key + 1] = '1';
                        output_file2 = fopen(string8, "w+b");
                        }
                }
        }
if (flag1 == 1)
        {
        if (data11 == 18 || data11 == 28 || data11 == 38)
                {
                if (l == 0)
                        {
                        key = 0;
                        i = 0;
                        while (string6[i] != '.' && i != 79)
                                {
                                i = i + 1;
                                if (string6[i] == '.') key = i;
                                }
                        string6[key + 1] = '1';
                        string6[key + 2] = 'y';
                        string6[key + 3] = 'y';
                        input_file1 = fopen(string6, "r+b");
                        for (mm = 0; mm < 80; mm++)
                                {
                                string7[mm] = 0;
                                }
                        string7[0] = 'Y';
                        string7[1] = '1';
                        output_file1 = fopen("Y1", "w+b");
                        }

                if (l == 1)
                        {
                        key = 0;
                        i = 0;
                        while (string6[i] != '.' && i != 79)
                                {
                                i = i + 1;
                                if (string6[i] == '.') key = i;
                                }
                        string6[key + 1] = '1';
                        string6[key + 2] = 'c';
                        string6[key + 3] = 'b';
                        input_file1 = fopen(string6, "r+b");
                        for (mm = 0; mm < 80; mm++)
                                {
                                string7[mm] = 0;
                                }
                        string7[0] = 'C';
                        string7[1] = 'b';
                        string7[2] = '1';
```

```
                output_file1 = fopen("Cb1", "w+b");
                }

        if (l == 2)
                {
                key = 0;
                i = 0;
                while (string6[i] != '.' && i != 79)
                        {
                        i = i + 1;
                        if (string6[i] == '.') key = i;
                        }
                string6[key + 1] = '1';
                string6[key + 2] = 'c';
                string6[key + 3] = 'r';
                input_file1 = fopen(string6, "r+b");
                for (mm = 0; mm < 80; mm++)
                        {
                        string7[mm] = 0;
                        }
                string7[0] = 'C';
                string7[1] = 'r';
                string7[2] = '1';
                output_file1 = fopen("Cr1", "w+b");
                }
        }
}

width4 = 32*xindex;
if (width4 != width3) width4 = width4 + 32;

if (extflag == 2) goto continue2;
if (extflag == 1) goto continue1;
if (extflag == 0) goto next;

continue3:
printf("\n.");
if (32 * yindex != height3) yindex = yindex + 1;
for( j = 0; j<=yindex-1; j++)
        {
        if (flag1 == 0)
        {
        printf(".");
        for ( y = 0; y<=31; y++)
                {
                for ( i = 0; i<=4; i++)
                        {
                        for ( x = 0; x<=31; x++)
                                {
                                if (j * 32 + (y + 1) > height3)
                                        {
```

```
                        data32[i][y][x] = 0;
                        }
                else
                        {
                        ykey = y;
                        fscanf(input_file1,"%c",&cp1);
                        data32[i][y][x] = cp1 - 128;
                        }
                }
        }
        x = 160;
        while (x != width3)
                {
                fscanf(input_file1,"%c",&cp1);
                x=x+1;
                }
        }
}
if (flag1 == 1)
{
printf(".");
for ( y = 0; y<=31; y++)
        {
        if (j * 32 + (y + 1) <= height3) ykey = y;
        for ( i = 0; i<=4; i++)
                {
                for ( x = 0; x<=31; x++)
                        {
                        fscanf(input_file1,"%c",&cp1);
                        data32[i][y][x] = cp1 - 128;
                        }
                }
                x = 160;
                while (x != width4)
                        {
                        fscanf(input_file1,"%c",&cp1);
                        x=x+1;
                        }
        }
}

if (flag1 == 0) doform32(5, data32, c32);
if (flag1 == 1) doinv32(5, data32, c32);

if (flag1 == 0)
{
for ( y = 0; y<=31; y++)
        {
        for ( i = 0; i<=4; i++)
                {
                for ( x = 0; x<=31; x++)
```



```
input_file1 = fopen(string8, "r+b");
```

```
                                {
                                if (data32[i][y][x] >= -threshold && data32[i][y][x]
<= threshold) data32[i][y][x] = 0;
                                if (data32[i][y][x] > 127) data32[i][y][x] = 127;
                                if (data32[i][y][x] < -128) data32[i][y][x] = -128;
                                cp1 = data32[i][y][x] + 128;
                                if (j * 32 + (y + 1) <= height3) putc (cp1,output_file1);
                                putc (cp1,output_file2);
                                }
                        }
                }
        }
        if (flag1 == 1)
        {
        for ( y = 0; y<=ykey; y++)
                {
                for ( i = 0; i<=4; i++)
                        {
                        for ( x = 0; x<=31; x++)
                                {
                                if (data32[i][y][x] > 127) data32[i][y][x] = 127;
                                if (data32[i][y][x] < -128) data32[i][y][x] = -128;
                                cp1 = data32[i][y][x] + 128;
                                putc (cp1,output_file1);
                                }
                         }
                }
        }
        }
fclose(input_file1);
fclose(output_file1);
if (flag1 == 0) fclose(output_file2);

xindex = width3/32;
yindex = height3/32;
input_file1 = fopen(string6,"r+b");
output_file1 = fopen("temp5","w+b");
if (flag1 == 0) output_file2 = fopen("temp6","w+b");


continue2:
printf("\n.");
if (32 * yindex != height3) yindex = yindex + 1;
for( j = 0; j<=yindex-1; j++)
        {
        if (flag1 == 0)
        {
        printf(".");
        for ( y = 0; y<=31; y++)
                {
                if (extflag == 3)
```

```
                {
                for (k = 0; k < 160; k++) fscanf(input_file1,"%c",&cp1);
                }
        for ( i = 0; i<=4; i++)
                {
                for ( x = 0; x<=31; x++)
                        {
                        if (j * 32 + (y + 1) > height3)
                                {
                                data32[i][y][x] = 0;
                                }
                        else
                                {
                                ykey = y;
                                fscanf(input_file1,"%c",&cp1);
                                data32[i][y][x] = cp1 - 128;
                                }
                        }
                }
                if (extflag == 2) x = 160;
                if (extflag == 3) x = 320;
                while (x != width3)
                        {
                        fscanf(input_file1,"%c",&cp1);
                        x=x+1;
                        }
        }
}
if (flag1 == 1)
{
printf(".");
for ( y = 0; y<=31; y++)
        {
        if (extflag == 3)
                {
                for (k = 0; k < 160; k++) fscanf(input_file1,"%c",&cp1);
                }
        if (j * 32 + (y + 1) <= height3) ykey = y;
        for ( i = 0; i<=4; i++)
                {
                for ( x = 0; x<=31; x++)
                        {
                        fscanf(input_file1,"%c",&cp1);
                        data32[i][y][x] = cp1 - 128;
                        }
                }
                if (extflag == 2) x = 160;
                if (extflag == 3) x = 320;
                while (x != width4)
                        {
                        fscanf(input_file1,"%c",&cp1);
```

```
                                x=x+1;
                                }
                        }
                }

                if (flag1 == 0) doform32(5, data32, c32);
                if (flag1 == 1) doinv32(5, data32, c32);

                if (flag1 == 0)
                {
                for ( y = 0; y<=31; y++)
                        {
                        for ( i = 0; i<=4; i++)
                                {
                                for ( x = 0; x<=31; x++)
                                        {
                                        if (data32[i][y][x] >= -threshold && data32[i][y][x]
        <= threshold) data32[i][y][x] = 0;
                                        if (data32[i][y][x] > 127) data32[i][y][x] = 127;
                                        if (data32[i][y][x] < -128) data32[i][y][x] = -128;
                                        cp1 = data32[i][y][x] + 128;
                                        if (j * 32 + (y + 1) <= height3) putc (cp1,output_file1);
                                        putc (cp1,output_file2);
                                        }
                                }
                        }
                }
                if (flag1 == 1)
                {
                for ( y = 0; y<=ykey; y++)
                        {
                        for ( i = 0; i<=4; i++)
                                {
                                for ( x = 0; x<=31; x++)
                                        {
                                        if (data32[i][y][x] > 127) data32[i][y][x] = 127;
                                        if (data32[i][y][x] < -128) data32[i][y][x] = -128;
                                        cp1 = data32[i][y][x] + 128;
                                        putc (cp1,output_file1);
                                        }
                                 }
                        }
                }
                }
        fclose(input_file1);
        fclose(output_file1);
        if (flag1 == 0) fclose(output_file2);

        xindex = width3/32;
        yindex = height3/32;
        input_file1 = fopen(string6,"r+b");
```

```
output_file1 = fopen("temp3","w+b");
if (flag1 == 0) output_file2 = fopen("temp4","w+b");

continue1:
printf("\n.");
if (32 * yindex != height3) yindex = yindex + 1;
for( j = 0; j<=yindex-1; j++)
        {
        if (flag1 == 0)
        {
        printf(".");
        for ( y = 0; y<=31; y++)
                {
                if (extflag == 2)
                        {
                        for (k = 0; k < 160; k++) fscanf(input_file1,"%c",&cp1);
                        }
                if (extflag == 3)
                        {
                        for (k = 0; k < 320; k++) fscanf(input_file1,"%c",&cp1);
                        }
                for ( i = 0; i<=4; i++)
                        {
                        for ( x = 0; x<=31; x++)
                                {
                                if (j * 32 + (y + 1) > height3)
                                        {
                                        data32[i][y][x] = 0;
                                        }
                                else
                                        {
                                        ykey = y;
                                        fscanf(input_file1,"%c",&cp1);
                                        data32[i][y][x] = cp1 - 128;
                                        }
                                }
                        }
                        if (extflag == 1) x = 160;
                        if (extflag == 2) x = 320;
                        if (extflag == 3) x = 480;
                        while (x != width3)
                                {
                                fscanf(input_file1,"%c",&cp1);
                                x=x+1;
                                }
                }
        }
        if (flag1 == 1)
        {
        printf(".");
        for ( y = 0; y<=31; y++)
```

```
                {
                if (extflag == 2)
                        {
                        for (k = 0; k < 160; k++) fscanf(input_file1,"%c",&cp1);
                        }
                if (extflag == 3)
                        {
                        for (k = 0; k < 320; k++) fscanf(input_file1,"%c",&cp1);
                        }
                if (j * 32 + (y + 1) <= height3) ykey = y;
                for ( i = 0; i<=4; i++)
                        {
                        for ( x = 0; x<=31; x++)
                                {
                                fscanf(input_file1,"%c",&cp1);
                                data32[i][y][x] = cp1 - 128;
                                }
                        }
                        if (extflag == 1) x = 160;
                        if (extflag == 2) x = 320;
                        if (extflag == 3) x = 480;
                        while (x != width4)
                                {
                                fscanf(input_file1,"%c",&cp1);
                                x=x+1;
                                }
                }
        }

        if (flag1 == 0) doform32(5, data32, c32);
        if (flag1 == 1) doinv32(5, data32, c32);

        if (flag1 == 0)
        {
        for ( y = 0; y<=31; y++)
                {
                for ( i = 0; i<=4; i++)
                        {
                        for ( x = 0; x<=31; x++)
                                {
                                if (data32[i][y][x] >= -threshold && data32[i][y][x]
<= threshold) data32[i][y][x] = 0;
                                if (data32[i][y][x] > 127) data32[i][y][x] = 127;
                                if (data32[i][y][x] < -128) data32[i][y][x] = -128;
                                cp1 = data32[i][y][x] + 128;
                                if (j * 32 + (y + 1) <= height3) putc (cp1,output_file1);
                                putc (cp1,output_file2);
                                }
                        }
                }
        }
```

```
        if (flag1 == 1)
        {
        for ( y = 0; y<=ykey; y++)
                {
                for ( i = 0; i<=4; i++)
                        {
                        for ( x = 0; x<=31; x++)
                                {
                                if (data32[i][y][x] > 127) data32[i][y][x] = 127;
                                if (data32[i][y][x] < -128) data32[i][y][x] = -128;
                                cp1 = data32[i][y][x] + 128;
                                putc (cp1,output_file1);
                                }
                         }
                }
        }
        }
fclose(input_file1);
fclose(output_file1);
if (flag1 == 0) fclose(output_file2);

xindex = width3/32;
yindex = height3/32;
input_file1 = fopen(string6,"r+b");
output_file1 = fopen("temp1","w+b");
if (flag1 == 0) output_file2 = fopen("temp2","w+b");
if (extflag == 1) nxindex = xindex - 5;
if (extflag == 2) nxindex = xindex - 10;
if (extflag == 3) nxindex = xindex - 15;

next:
if (extflag == 0) nxindex = xindex;
printf("\n.");
if (32 * yindex != height3) yindex = yindex + 1;
for( j = 0; j<=yindex-1; j++)
        {
        if (flag1 == 0)
        {
        printf(".");
        for ( y = 0; y<=31; y++)
                {
                if (extflag == 1)
                        {
                        for (k = 0; k < 160; k++) fscanf(input_file1,"%c",&cp1);
                        }
                if (extflag == 2)
                        {
                        for (k = 0; k < 320; k++) fscanf(input_file1,"%c",&cp1);
                        }
                if (extflag == 3)
                        {
```

```
                for (k = 0; k < 480; k++) fscanf(input_file1,"%c",&cp1);
                }
        for ( i = 0; i<=nxindex-1; i++)
                {
                for ( x = 0; x<=31; x++)
                        {
                        if (j * 32 + (y + 1) > height3)
                                {
                                data32[i][y][x] = 0;
                                }
                        else
                                {
                                ykey = y;
                                fscanf(input_file1,"%c",&cp1);
                                data32[i][y][x] = cp1 - 128;
                                }
                        }
                }
                if ( 32 * xindex != width3)
                        {
                        xkey = 32 * xindex;
                        i = nxindex;
                        x = 0;
                        while (xkey != width3)
                                {
                                if (j * 32 + (y + 1) > height3)
                                        {
                                        data32[i][y][x] = 0;
                                        }
                                else
                                        {
                                        fscanf(input_file1,"%c",&cp1);
                                        data32[i][y][x] = cp1 - 128;
                                        }
                                x = x + 1;
                                xkey = xkey + 1;
                                }
                        while (x != 32)
                                {
                                data32[i][y][x] = 0;
                                x = x + 1;
                                }
                        }
        }
}
if (flag1 == 1)
{
printf(".");
for ( y = 0; y<=31; y++)
        {
        if (extflag == 1)
```

```
                {
                for (k = 0; k < 160; k++) fscanf(input_file1,"%c",&cp1);
                }
        if (extflag == 2)
                {
                for (k = 0; k < 320; k++) fscanf(input_file1,"%c",&cp1);
                }
        if (extflag == 3)
                {
                for (k = 0; k < 480; k++) fscanf(input_file1,"%c",&cp1);
                }
        if (j * 32 + (y + 1) <= height3) ykey = y;
        for ( i = 0; i<=nxindex-1; i++)
                {
                for ( x = 0; x<=31; x++)
                        {
                        fscanf(input_file1,"%c",&cp1);
                        data32[i][y][x] = cp1 - 128;
                        }
                }
                if ( 32 * xindex != width3)
                        {
                        xkey = 32 * xindex;
                        i = nxindex;
                        x = 0;
                        while (xkey != width3)
                                {
                                fscanf(input_file1,"%c",&cp1);
                                data32[i][y][x] = cp1 - 128;
                                x = x + 1;
                                xkey = xkey + 1;
                                }
                        while (x != 32)
                                {
                                fscanf(input_file1,"%c",&cp1);
                                data32[i][y][x] = cp1 - 128;
                                x = x + 1;
                                }
                        }
        }
}

ixindex = nxindex;
if (32 * xindex != width3) ixindex = ixindex + 1;
if (flag1 == 0) doform32(ixindex, data32, c32);
if (flag1 == 1) doinv32(ixindex, data32, c32);

if (flag1 == 0)
{
for ( y = 0; y<=31; y++)
        {
```

```
                for ( i = 0; i<=nxindex-1; i++)
                        {
                        for ( x = 0; x<=31; x++)
                                {
                                if (data32[i][y][x] >= -threshold && data32[i][y][x]
<= threshold) data32[i][y][x] = 0;
                                if (data32[i][y][x] > 127) data32[i][y][x] = 127;
                                if (data32[i][y][x] < -128) data32[i][y][x] = -128;
                                cp1 = data32[i][y][x] + 128;
                                if (j * 32 + (y + 1) <= height3) putc (cp1,output_file1);
                                putc (cp1,output_file2);
                                }
                        }
                        if ( 32 * xindex != width3)
                                {
                                xkey = 32 * xindex;
                                i = nxindex;
                                x = 0;
                                while (xkey != width3)
                                        {
                                        if (data32[i][y][x] >= -threshold && data32[i][y][x]
<= threshold) data32[i][y][x] = 0;
                                        if (data32[i][y][x] > 127) data32[i][y][x] = 127;
                                        if (data32[i][y][x] < -128) data32[i][y][x] = -128;
                                        cp1 = data32[i][y][x] + 128;
                                        if (j * 32 + (y + 1) <= height3) putc (cp1,output_file1);
                                        putc (cp1,output_file2);
                                        x = x + 1;
                                        xkey = xkey + 1;
                                        }
                                while (x != 32)
                                        {
                                        if (data32[i][y][x] >= -threshold && data32[i][y][x]
<= threshold) data32[i][y][x] = 0;
                                        if (data32[i][y][x] > 127) data32[i][y][x] = 127;
                                        if (data32[i][y][x] < -128) data32[i][y][x] = -128;
                                        cp1 = data32[i][y][x] + 128;
                                        putc (cp1,output_file2);
                                        x = x + 1;
                                        }
                                 }
                }
        }
        if (flag1 == 1)
        {
        for ( y = 0; y<=ykey; y++)
                {
                for ( i = 0; i<=nxindex-1; i++)
                        {
                        for ( x = 0; x<=31; x++)
                                {
```

```
                        if (data32[i][y][x] > 127) data32[i][y][x] = 127;
                        if (data32[i][y][x] < -128) data32[i][y][x] = -128;
                        cp1 = data32[i][y][x] + 128;
                        putc (cp1,output_file1);
                        }
                }
                if ( 32 * xindex != width3)
                        {
                        xkey = 32 * xindex;
                        i = nxindex;
                        x = 0;
                        while (xkey != width3)
                                {
                                if (data32[i][y][x] > 127) data32[i][y][x] = 127;
                                if (data32[i][y][x] < -128) data32[i][y][x] = -128;
                                cp1 = data32[i][y][x] + 128;
                                putc (cp1,output_file1);
                                x = x + 1;
                                xkey = xkey + 1;
                                }
                        }
        }
        }
        }
fclose(input_file1);
fclose(output_file1);
if (flag1 == 0) fclose(output_file2);


if (extflag == 0 && zone != 32)
{
if (flag1 == 0)
{
input_file1 = fopen(string8, "r+b");
output_file1 = fopen("temp1", "w+b");

if (xindex*32 != width3) xindex = xindex + 1;

for (y = 0; y < yindex; y++)
        {
        for (i = 0; i < 32; i++)
        {
        for (x = 0; x < xindex; x++)
                {
                for (j = 0; j < 32; j++)
                {
                fscanf(input_file1,"%c",&cp1);
                putc (cp1,output_file1);
                }
                }
        }
```

```
        }

fclose(input_file1);
fclose(output_file1);

input_file1 = fopen("temp1", "r+b");
output_file1 = fopen(string8, "w+b");

for (y = 0; y < yindex; y++)
        {
        for (i = 0; i < 32; i++)
        {
        for (x = 0; x < xindex; x++)
                {
                for (j = 0; j < 32; j++)
                {
                fscanf(input_file1,"%c",&cp1);
                if ( i >= zone || j >= zone) cp1 = 128;
                putc (cp1,output_file1);
                }
                }
        }
        }

fclose(input_file1);
fclose(output_file1);
}
}

if (extflag == 1)
{
input_file1 = fopen(string7, "r+b");
input_file2 = fopen("temp1", "r+b");
output_file1 = fopen("temp3", "w+b");

for (y = 0; y < height3; y++)
        {
        for (x = 0; x < width3; x++)
                {
                if (x < 160) fscanf(input_file1,"%c",&cp1);
                else fscanf(input_file2,"%c",&cp1);
                putc (cp1,output_file1);
                }
        }
fclose(input_file1);
fclose(input_file2);
fclose(output_file1);

input_file1 = fopen("temp3", "r+b");
output_file1 = fopen(string7, "w+b");
```

```
for (y = 0; y < height3; y++)
        {
        for (x = 0; x < width3; x++)
                {
                fscanf(input_file1,"%c",&cp1);
                putc (cp1,output_file1);
                }
        }

fclose(input_file1);
fclose(output_file1);

if (flag1 == 0)
{
input_file1 = fopen(string8, "r+b");
input_file2 = fopen("temp2", "r+b");
output_file1 = fopen("temp3", "w+b");


if (xindex*32 != width3) xindex = xindex + 1;

for (y = 0; y < 32*yindex; y++)
        {
        for (x = 0; x < 32*xindex; x++)
                {
                if (x < 160) fscanf(input_file1,"%c",&cp1);
                else fscanf(input_file2,"%c",&cp1);
                putc (cp1,output_file1);
                }
        }
fclose(input_file1);
fclose(input_file2);
fclose(output_file1);

input_file1 = fopen("temp3", "r+b");
output_file1 = fopen(string8, "w+b");

for (y = 0; y < yindex; y++)
        {
        for (i = 0; i < 32; i++)
        {
        for (x = 0; x < xindex; x++)
                {
                for (j = 0; j < 32; j++)
                {
                fscanf(input_file1,"%c",&cp1);
                if ( i >= zone || j >= zone) cp1 = 128;
                putc (cp1,output_file1);
                }
                }
        }
```

```
        }

fclose(input_file1);
fclose(output_file1);
}

remove("temp1");
remove("temp2");
remove("temp3");

}

if (extflag == 2)
{
input_file1 = fopen(string7, "r+b");
input_file2 = fopen("temp3", "r+b");
input_file3 = fopen("temp1", "r+b");
output_file1 = fopen("temp5", "w+b");

for (y = 0; y < height3; y++)
        {
        for (x = 0; x < width3; x++)
                {
                if (x < 160) fscanf(input_file1,"%c",&cp1);
                else if (x < 320) fscanf(input_file2,"%c",&cp1);
                else fscanf(input_file3,"%c",&cp1);
                putc (cp1,output_file1);
                }
        }
fclose(input_file1);
fclose(input_file2);
fclose(input_file3);
fclose(output_file1);

input_file1 = fopen("temp5", "r+b");
output_file1 = fopen(string7, "w+b");

for (y = 0; y < height3; y++)
        {
        for (x = 0; x < width3; x++)
                {
                fscanf(input_file1,"%c",&cp1);
                putc (cp1,output_file1);
                }
        }

fclose(input_file1);
fclose(output_file1);

if (flag1 == 0)
{
```

```
input_file1 = fopen(string8, "r+b");
input_file2 = fopen("temp4", "r+b");
input_file3 = fopen("temp2", "r+b");
output_file1 = fopen("temp5", "w+b");

if (xindex*32 != width3) xindex = xindex + 1;

for (y = 0; y < yindex*32; y++)
        {
        for (x = 0; x < xindex*32; x++)
                {
                if (x < 160) fscanf(input_file1,"%c",&cp1);
                else if (x < 320) fscanf(input_file2,"%c",&cp1);
                else fscanf(input_file3,"%c",&cp1);
                putc (cp1,output_file1);
                }
        }
fclose(input_file1);
fclose(input_file2);
fclose(input_file3);
fclose(output_file1);

input_file1 = fopen("temp5", "r+b");
output_file1 = fopen(string8, "w+b");

for (y = 0; y < yindex; y++)
        {
        for (i = 0; i < 32; i++)
        {
        for (x = 0; x < xindex; x++)
                {
                for (j = 0; j < 32; j++)
                {
                fscanf(input_file1,"%c",&cp1);
                if ( i >= zone || j >= zone) cp1 = 128;
                putc (cp1,output_file1);
                }
                }
        }
        }

fclose(input_file1);
fclose(output_file1);
}

remove("temp1");
remove("temp2");
remove("temp3");
remove("temp4");
remove("temp5");
}
```

```
if (extflag == 3)
{
input_file1 = fopen(string7, "r+b");
input_file2 = fopen("temp5", "r+b");
input_file3 = fopen("temp3", "r+b");
input_file4 = fopen("temp1", "r+b");
output_file1 = fopen("temp7", "w+b");

for (y = 0; y < height3; y++)
        {
        for (x = 0; x < width3; x++)
                {
                if (x < 160) fscanf(input_file1,"%c",&cp1);
                else if (x < 320) fscanf(input_file2,"%c",&cp1);
                else if (x < 480) fscanf(input_file3,"%c",&cp1);
                else fscanf(input_file4,"%c",&cp1);
                putc (cp1,output_file1);
                }
        }
fclose(input_file1);
fclose(input_file2);
fclose(input_file3);
fclose(input_file4);
fclose(output_file1);

input_file1 = fopen("temp7", "r+b");
output_file1 = fopen(string7, "w+b");

for (y = 0; y < height3; y++)
        {
        for (x = 0; x < width3; x++)
                {
                fscanf(input_file1,"%c",&cp1);
                putc (cp1,output_file1);
                }
        }

fclose(input_file1);
fclose(output_file1);

if (flag1 == 0)
{
input_file1 = fopen(string8, "r+b");
input_file2 = fopen("temp6", "r+b");
input_file3 = fopen("temp4", "r+b");
input_file4 = fopen("temp2", "r+b");
output_file1 = fopen("temp7", "w+b");

if (xindex*32 != width3) xindex = xindex + 1;
```

```
for (y = 0; y < yindex*32; y++)
        {
        for (x = 0; x < xindex*32; x++)
                {
                if (x < 160) fscanf(input_file1,"%c",&cp1);
                else if (x < 320) fscanf(input_file2,"%c",&cp1);
                else if (x < 480) fscanf(input_file3,"%c",&cp1);
                else fscanf(input_file4,"%c",&cp1);
                putc (cp1,output_file1);
                }
        }
fclose(input_file1);
fclose(input_file2);
fclose(input_file3);
fclose(input_file4);
fclose(output_file1);

input_file1 = fopen("temp7", "r+b");
output_file1 = fopen(string8, "w+b");

for (y = 0; y < yindex; y++)
        {
        for (i = 0; i < 32; i++)
        {
        for (x = 0; x < xindex; x++)
                {
                for (j = 0; j < 32; j++)
                {
                fscanf(input_file1,"%c",&cp1);
                if ( i >= zone || j >= zone) cp1 = 128;
                putc (cp1,output_file1);
                }
                }
        }
        }

 fclose(input_file1);
 fclose(output_file1);
 }

 remove("temp1");
 remove("temp2");
 remove("temp3");
 remove("temp4");
 remove("temp5");
 remove("temp6");
 remove("temp7");
 }
 }

 if (data11 == 18 || data11 == 28 || data11 == 38)
```

```
{
if (flag1 == 1)
{
input_file1 = fopen("Y1", "r+b");
input_file2 = fopen("Cb1", "r+b");
input_file3 = fopen("Cr1", "r+b");
output_file1 = fopen(string9, "w+b");
while (!feof(input_file1) && !feof(input_file2) && !feof(input_file3))
        {
        fscanf (input_file1,"%c",&cp1);
        fscanf (input_file2,"%c",&cp2);
        fscanf (input_file3,"%c",&cp3);
        interm = cp1;
        interm2 = interm;
        interm = cp3 - 128;
        interm1 = interm;
        interm2 = interm2 + interm1 * 1.402;
        interm = (interm2 + .5);
        if (interm < 0) interm = 0;
        cp4 = interm;
        if (cp4 > 208) cp4 = 224;
        else if (cp4 > 176) cp4 = 192;
                else if (cp4 > 144) cp4 = 160;
                        else if (cp4 > 112) cp4 = 128;
                                else if (cp4 > 80) cp4 = 96;
                                        else if (cp4 > 48) cp4 = 64;
                                                else if (cp4 > 16) cp4 = 32;
                                                        else cp4 = 0;
        interm = cp1;
        interm2 = interm;
        interm = cp2 - 128;
        interm1 = interm;
        interm1 = interm1 * .34414;
        interm2 = interm2 - interm1;
        interm = cp3 - 128;
        interm1 = interm;
        interm1 = interm1 * .71414;
        interm2 = interm2 - interm1;
        interm = (interm2 + .5);
        if (interm < 0) interm = 0;
        cp5 = interm;
        if (cp5 > 208) cp5 = 224;
        else if (cp5 > 176) cp5 = 192;
                else if (cp5 > 144) cp5 = 160;
                        else if (cp5 > 112) cp5 = 128;
                                else if (cp5 > 80) cp5 = 96;
                                        else if (cp5 > 48) cp5 = 64;
                                                else if (cp5 > 16) cp5 = 32;
                                                        else cp5 = 0;
        cp5 = cp5 / 8;
        interm = cp1;
```

```
                interm2 = interm;
                interm = cp2 - 128;
                interm1 = interm;
                interm2 = interm2 + interm1 * 1.772;
                interm = (interm2 + .5);
                if (interm < 0) interm = 0;
                cp6 = interm;
                if (cp6 > 160) cp6 = 192;
                else if (cp6 > 96) cp6 = 128;
                        else if (cp6 > 32) cp6 = 64;
                                else cp6 = 0;
                cp6 = cp6 / 64;
                cp1 = cp4 + cp5 + cp6;
                putc (cp1,output_file1);
                }
        fclose(input_file1);
        fclose(input_file2);
        fclose(input_file3);
        fclose(output_file1);
        remove("Y1");
        remove("Cb1");
        remove("Cr1");
        }
        if (flag1 == 0)
        {
        remove("Y");
        remove("Cb");
        remove("Cr");
        }
        }

end1:
printf("");
}
```

```
#include <stdio.h>
#include <math.h>

/************************** START ANALYZE *************************/
void analyze(flag2)
int flag2;
{
float ratio1,count3=0.0,count4=0.0;
int i,k,x,y,xindex,yindex,diff,threshold,ch14,width2,
    height2,width3,height3,x1,x2,y1,y2;
char string[15],string1[15],string2[15],string3[15];
unsigned char p,cp1,cp2;
long int count=0L,count1[256],constant=1L,constant1=0L;
float j[256],pixel,entropy;

FILE *input_file;
FILE *infile;
FILE *infile1;
FILE *infile2;
FILE *outfile1;

Set_Mode(3);

if (flag2 == 0) goto entropy;
if (flag2 == 1) goto compare;
if (flag2 == 2) goto mse;
if (flag2 == 3) goto pixel;
if (flag2 == 4) goto ratio;
if (flag2 == 5) goto chop;
if (flag2 == 6) goto paste;
if (flag2 == 7) goto pastevert;
if (flag2 == 8) goto diffimage;

entropy:
printf("ENTROPY CALCULATION\n\n");
entropy = 0;
for ( i = 0; i<=255; i++)
        {
        j[i]=0;
        }
        pixel = 0;
restart10:
printf ("Type name of file: ");
scanf ("%s",string );
input_file=fopen(string,"r+b");
if (input_file == (FILE *) NULL)
        {
```

```
          printf("\n Bad file - try again  \n \n");
          goto restart10;
        }
while (!feof(input_file))
        {
        fscanf (input_file,"%c",&p);
        pixel = pixel + 1;
        k=p;
        j[k]=j[k]+1;
        }

for (i = 0; i <=255; i++)
        {
        if (j[i] !=0)
                {
                entropy = entropy + j[i]*log(j[i]/pixel);
                }
        }
        entropy = -entropy/(pixel*log(2));
        printf ("\nEntropy = %f",entropy);
        printf("\nHuffman encoding can achieve");
        printf(" a %f to 1 compression.\n",8/entropy);
getch();
goto end2;

compare:
printf("COMPARE FILES\n\n");
restart11:
printf("Input file 1: ");
scanf("%s",string1);
infile1 = fopen(string1,"r+b");
if (infile1 == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart11;
        }
restart12:
printf("Input file 2: ");
scanf("%s",string2);
infile2 = fopen(string2,"r+b");
if (infile2 == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart12;
        }
printf("Width: ");
scanf("%i",&xindex);
printf("Height: ");
scanf("%i",&yindex);
printf("Threshold difference: ");
scanf("%i",&threshold);
```

```
for (y = 0; y < yindex; y++)
        {
        for (x = 0; x < xindex; x++)
                {
                fscanf(infile1,"%c",&cp1);
                fscanf(infile2,"%c",&cp2);
                diff = cp1 - cp2;
                if (diff < 0) diff = diff - 2 * diff;
                if (diff >= threshold)
                        {
                        printf("\nPixel 1 = %3i, Pixel 2 = %3i, x = %3i, y = %3i",cp1,cp2,x,y);
                        ch14 = getch();
                        if (ch14 == 'q' || ch14 == 'Q') goto end2;
                        }
                }
        }
fclose(infile1);
fclose(infile2);
goto end2;

pixel:
printf("HISTOGRAM\n\n");
restart13:
printf("Input file: ");
scanf("%s",string1);
infile1 = fopen(string1,"r+b");
if (infile1 == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart13;
        }
for (i = 0; i < 256; i++) count1[i] = constant1;
while (!feof(infile1))
        {
        fscanf(infile1,"%c",&cp1);
        count = count + constant;
        x = cp1;
        count1[x] = count1[x] + constant;
        }
printf("\n\nTotal = %li\n",count-1);
for (y = 0; y < 16; y++)
        {
        for (x = 0; x < 16; x++)
                {
                i = y * 16 + x;
                if (count1[i] != 1) printf("\nPixel %3i occurs %6li times.",i,count1[i]);
                else printf("\nPixel %3i occurs %6li time.",i,count1[i]);
                }
        getch();
        }
```

```
fclose(infile1);
goto end2;

ratio:
printf("COMPRESSION RATIO CALCULATION\n\n");
restart14:
printf("Original file: ");
scanf("%s",string1);
infile1 = fopen(string1,"r+b");
if (infile1 == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart14;
        }

restart15:
printf("Compressed file: ");
scanf("%s",string2);
infile2 = fopen(string2,"r+b");
if (infile2 == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart15;
        }
while (!feof(infile1))
        {
        fscanf(infile1,"%c",&cp1);
        count3= count3 + constant;
        }
while (!feof(infile2))
        {
        fscanf(infile2,"%c",&cp1);
        count4 = count4 + constant;
        }
printf("\nOriginal Size = %.0f",count3-1);
printf("\nCompressed Size = %.0f",count4-1);
ratio1 = count3 / count4;
printf("\nCompression Achieved => 1:%.3f",ratio1);
getch();

fclose(infile1);
fclose(infile2);

goto end2;

mse:
printf("MEAN SQUARED ERROR CALCULATION\n\n");
restart16:
printf("Input file 1: ");
scanf("%s",string1);
infile1 = fopen(string1,"r+b");
```

```
if (infile1 == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart16;
        }
restart17:
printf("Input file 2: ");
scanf("%s",string2);
infile2 = fopen(string2,"r+b");
if (infile2 == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart17;
        }
while (!feof(infile1) && !feof(infile2))
        {
        fscanf(infile1,"%c",&cp1);
        fscanf(infile2,"%c",&cp2);
        diff = cp1 - cp2;
        if (diff < 0) diff = diff - 2 * diff;
        count = count + constant;
        count3 = count3 + diff * diff;
        }
printf("\nMean Squared Error = %2f",count3/count);
getch();

fclose(infile1);
fclose(infile2);
goto end2;

chop:
printf("CHOP IMAGE FILE\n\n");
restart21:
printf("Type name of input file: ");
scanf("%s",string1);
infile1 = fopen(string1,"r+b");
if (infile1 == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart21;
        }
printf("Enter width: ");
scanf("%i",&width2);
printf("Enter height: ");
scanf("%i",&height2);
printf("Type name of output file: ");
scanf("%s",string2);
outfile1 = fopen(string2,"w+b");
printf("Enter starting X coordinate: ");
scanf("%i",&x1);
printf("Enter ending X coordinate: ");
```

```
scanf("%i",&x2);
printf("Enter starting Y coordinate: ");
scanf("%i",&y1);
printf("Enter ending Y coordinate: ");
scanf("%i",&y2);

for (y = 1; y <= y2; y++)
        {
        for (x = 1; x <= width2; x++)
                {
                fscanf(infile1,"%c",&cp1);
                if(x >= x1 && x <= x2 && y >= y1 && y <= y2) putc(cp1,outfile1);
                }
        }

fclose(infile1);
fclose(outfile1);
goto end2;


paste:
printf("PASTE IMAGE FILES: HORIZONTALLY\n\n");
restart22:
printf("Type name of input file 1: ");
scanf("%s", string1);
infile = fopen(string1,"r+b");
if (infile == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart22;
        }
printf("Enter width: ");
scanf("%i", &width2);
printf("Enter height: ");
scanf("%i", &height2);
restart23:
printf("Type name of input file 2: ");
scanf("%s", string2);
infile1 = fopen(string2,"r+b");
if (infile1 == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart23;
        }
printf("Enter width: ");
scanf("%i", &width3);
printf("Enter height: ");
scanf("%i", &height3);

printf("Type name of output file: ");
scanf("%s", string3);
```

```
outfile1 = fopen(string3,"w+b");

xindex = width2 + width3;
if (height3 > height2) yindex = height3;
else yindex = height2;

for (k = 0; k < yindex; k++)
        {
        for (i = 0; i < xindex; i++)
        {
        if (height3 == height2)
                {
                if (i < width2) fscanf(infile,"%c",&cp1);
                else fscanf(infile1,"%c",&cp1);
                putc(cp1,outfile1);
                }
        if (height3 > height2)
                {
                if (i < width2 && k < height2) fscanf(infile,"%c",&cp1);
                if (i < width2 && k >= height2) cp1 = 0;
                if (i >= width2) fscanf(infile1,"%c",&cp1);
                putc(cp1,outfile1);
                }
        if (height3 < height2)
                {
                if (i < width2) fscanf(infile,"%c",&cp1);
                if (i >= width2 && k < height3) fscanf(infile1,"%c",&cp1);
                if (i >= width2 && k >= height3) cp1 = 0;
                putc(cp1,outfile1);
                }
        }
        }
fclose(infile);
fclose(infile1);
fclose(outfile1);
goto end2;

pastevert:
printf("PASTE IMAGE FILES: VERTICALLY\n\n");
restart50:
printf("Type name of input file 1: ");
scanf("%s", string1);
infile = fopen(string1,"r+b");
if (infile == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart50;
        }
printf("Enter width: ");
scanf("%i", &width2);
printf("Enter height: ");
```

```
scanf("%i", &height2);
restart51:
printf("Type name of input file 2: ");
scanf("%s", string2);
infile1 = fopen(string2,"r+b");
if (infile == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart51;
        }
printf("Enter width: ");
scanf("%i", &width3);
printf("Enter height: ");
scanf("%i", &height3);

printf("Type name of output file: ");
scanf("%s", string3);
outfile1 = fopen(string3,"w+b");

if (width3 > width2) xindex = width3;
else xindex = width2;
yindex = height2 + height3;

for (k = 0; k < yindex; k++)
        {
        for (i = 0; i < xindex; i++)
        {
        if (width3 == width2)
                {
                if (k < height2) fscanf(infile,"%c",&cp1);
                else fscanf(infile1,"%c",&cp1);
                putc(cp1,outfile1);
                }
        if (width3 > width2)
                {
                if (k < height2 && i < width2) fscanf(infile,"%c",&cp1);
                if (k < height2 && i >= width2) cp1 = 0;
                if (k >= height2) fscanf(infile1,"%c",&cp1);
                putc(cp1,outfile1);
                }
        if (width3 < width2)
                {
                if (k < height2) fscanf(infile,"%c",&cp1);
                if (k >= height2 && i < width3) fscanf(infile1,"%c",&cp1);
                if (k >= height2 && i >= width3) cp1 = 0;
                putc(cp1,outfile1);
                }
        }
        }

fclose(infile);
```

```
fclose(infile1);
fclose(outfile1);
goto end2;

diffimage:
printf("DIFFERENCE IMAGES\n\n");
restart52:
printf("Original file: ");
scanf("%s",string1);
infile1 = fopen(string1,"r+b");
if (infile == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart52;
        }
restart53:
printf("Reconstructed file: ");
scanf("%s",string2);
infile2 = fopen(string2,"r+b");
if (infile == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart53;
        }
printf("Width: ");
scanf("%i",&xindex);
printf("Height: ");
scanf("%i",&yindex);
printf("Output file: ");
scanf("%s",string3);
outfile1 = fopen(string3,"w+b");
for (y = 0; y < yindex; y++)
        {
        for (x = 0; x < xindex; x++)
                {
                fscanf(infile1,"%c",&cp1);
                fscanf(infile2,"%c",&cp2);
                diff = cp1 - cp2;
                if (diff > 26) cp1 = 255;
                if (diff < -26) cp1 = 0;
                else cp1 = 128 + diff * 5;
                putc(cp1,outfile1);
                }
        }
fclose(infile1);
fclose(infile2);
fclose(outfile1);
goto end2;

end2:
printf("");
```

```
}
/***************************** END ANALYZE ****************************/
```

Appendix F
Source Code for "Image View" Software

IMAGVIEW.BAT

```
cl /c /DM5 /AL /I\cscape\include view.c
link /stack:44000 /SE:300 view,,,\global\global+\standard\standard
+\cscape\lib\m1lcscap+\cscape\lib\m1low1;
```

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <dos.h>
#include <stdarg.h>
#include <ctype.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>
#include "errhand.h"
#include "bitio.h"
#include "\global\globdef.h"
#include "\cscape\include\cscape.h"
#include "\cscape\include\framer.h"


 int vga_code,graphics,width,height,mode,modes[5],choice,
     read_bank,write_bank,top,stat_buf[4],radius,ix,iy,count;
 char instring1[80],message_string[81],string[80],string3[20];
 unsigned stringpal1[768],colorflag=2;
 sed_type frame;

 FILE *inpalette;
 FILE *input_file;
 FILE *input_file1;


void graphics1(idata, mask, ch)
int idata,mask,ch;
{
int i,j,p,l,m,ii,jj,kk,mm,red2[8],green2[8],blue2[8],x,y;
unsigned char red[256],green[256],blue[256],red1[256],green1[256],
    blue1[256],point_color,cp,ch3;

FILE *input_file;
FILE *inpalette;
FILE *out;

if (idata == 21) goto loadimage;
if (ch != 0) goto preimage;

/*****************************IMAGE************************/
loadimage:
Set_Mode(3);
restart19:
printf ("Type name of image series file: ");
scanf ("%s",string);
```

```
input_file=fopen(string,"r+b");
if (input_file == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart19;
        }
fscanf (input_file,"%i",&count);
goto image;

preimage:
input_file=fopen(string,"r+b");
fscanf (input_file,"%i",&count);

image:
/************************VIDEO MODES*********************/
setmode:
Set_Mode(3);
fscanf(input_file,"%i",&ch3);
if (ch3 == 1)
        {
        width = 320;
        mode = modes[1];
        }
if (ch3 == 2)
        {
        width = 640;
        mode = modes[2];
        }
if (ch3 == 3)
        {
        width = 800;
        mode = modes[3];
        }
if (ch3 == 4)
        {
        width = 1024;
        mode = modes[4];
        }
if (mode == -1)
        printf ("\nError");
if(Number_Color==256)
 { Load_Write_Bank_256(0); Load_Read_Bank_256(0); }
if(Number_Color==16)
 { Load_Write_Bank_16(0);  Load_Read_Bank_16(0);  }
/************************VIDEO MODES*********************/

/************************LOAD PALETTE*********************/
loadpalette:
restart18:
fscanf (input_file,"%s",instring1);
inpalette=fopen(instring1,"r+b");
```

```
if (inpalette == (FILE *) NULL)
        {
          printf("\n Bad file - try again  \n \n");
          goto restart18;
        }
p = 0;
while (!feof(inpalette))
        {
        fscanf (inpalette,"%c",&stringpal1[p]);
        p = p + 1;
        }
fclose(inpalette);
/************************LOAD PALETTE*********************/

count = count - 1;
fscanf (input_file,"%s",string3);

input_file1=fopen(string3,"r+b");
fscanf (input_file,"%i",&ix);
fscanf (input_file,"%i",&iy);

if (ch != 0) input_file1=fopen(string3,"r+b");
Set_Mode(mode);
for ( ii = 0; ii<=255; ii++)
        {
        red[ii] = stringpal1[ii];
        }
for ( jj = 256; jj<=511; jj++)
        {
        l = jj - 256;
        green[l] = stringpal1[jj];
        }
for ( kk = 512; kk<=767; kk++)
        {
        m = kk - 512;
        blue[m] = stringpal1[kk];
        }
outp(0x3c8,0);
for ( mm = 0; mm<=255; mm++)
         {
         if (ch == 0 || ch == 1 || ch == 4 || ch == 6) outp(0x3c9,red[mm]/4); else outp(0x3c9,0);
         if (ch == 0 || ch == 2 || ch == 4 || ch == 6) outp(0x3c9,green[mm]/4); else outp(0x3c9,0);
         if (ch == 0 || ch == 3 || ch == 4 || ch == 6) outp(0x3c9,blue[mm]/4); else outp(0x3c9,0);
         }
for (y=0; y<=iy-1;y++)
        {
        for (x=0; x<=ix-1;x++)
              {
              if (y == 200 && width == 320) goto end9;
              if (y == 480 && width == 640) goto end9;
              if (y == 600 && width == 800) goto end9;
```

```
                if (y == 768 && width == 1024) goto end9;
                fscanf (input_file1, "%c",&cp);
                if (ch == 4)
                    {
                    point_color = cp & mask;
                    if (point_color && ch == 4) point_color = 255;
                    }
                 else point_color = cp;
                 if (x < 1024) WrPixel_256(x,y,point_color,width);
                 }
        }
        end9:
        fclose(input_file1);
        getch();

end4:
if (count != 0) goto image;
fclose(input_file);
}
/*****************************IMAGE************************/

int mmode(idata)
int idata;
{
int ch30;

Set_Mode(3);
printf("AVAILABLE MENU COLORS\n");
printf("\n1.  Red");
printf("\n2.  Green");
printf("\n3.  Blue");
printf("\n4.  B&W");
printf("\n\nSelection: ");
ch30 = getch();
printf("%c",ch30);
getch();
if (ch30 == '1') colorflag = 0;
if (ch30 == '2') colorflag = 1;
if (ch30 == '3') colorflag = 2;
if (ch30 == '4') colorflag = 3;
Set_Mode(3);
return(1);
}

int metro(sdata, data)
char *sdata;
int data;
{
int mask,ch;
mask = 255;
ch = 0;
```

```
if (data == 23) ch = 1;
if (data == 24) ch = 2;
if (data == 25) ch = 3;
if (data == 26)
        {
        ch = 4;
        mask = 128;
        }
if (data == 27)
        {
        ch = 4;
        mask = 64;
        }
if (data == 28)
        {
        ch = 4;
        mask = 32;
        }
if (data == 29)
        {
        ch = 4;
        mask = 16;
        }
if (data == 30)
        {
        ch = 4;
        mask = 8;
        }
if (data == 31)
        {
        ch = 4;
        mask = 4;
        }
if (data == 32)
        {
        ch = 4;
        mask = 2;
        }
if (data == 33)
        {
        ch = 4;
        mask = 1;
        }
if (data == 34) ch = 6;
graphics1(data,mask,ch);
Set_Mode(3);
return(1);
}

void getout(idata)
int idata;
```

```
{
Set_Mode(3);
exit(1);
}

 static struct frame_def main_frame[] = {

 { "FILE  ", NULL, 10},

 { "Load Series File", metro, 21},
 { "Quit", getout, 0},
 { FRAME_END },

 { "VIEW  ", NULL, 11},

 { "Show Red", metro, 23},
 { "Show Green", metro, 24},
 { "Show Blue", metro, 25},
 { "Show Bit Plane 7", metro, 26},
 { "Show Bit Plane 6", metro, 27},
 { "Show Bit Plane 5", metro, 28},
 { "Show Bit Plane 4", metro, 29},
 { "Show Bit Plane 3", metro, 30},
 { "Show Bit Plane 2", metro, 31},
 { "Show Bit Plane 1", metro, 32},
 { "Show Bit Plane 0", metro, 33},
 { "Display Full Image", metro, 34},
 { FRAME_END },

 { "OPTIONS", NULL, 15},

 { "Set Menu Color", mmode, 0},
 { FRAME_END },
 { FRAME_END }
 };

/*************************************************************************/
/*************************************************************************/
main()
{
 int i,j,k,l,m,p,ii,jj,kk,mm,red[256],green[256],blue[256];

 FILE *inpalette;

 void getout(void);
 int mmode(int idata);
 int metro(char *sdata, int data);
 void graphics1(int idata, int mask, int ch);

/*****************************MENU*********************************/
Build_Mode();
```

```
vga_code=Which_VGA();
VGA_id = vga_code;
if(vga_code<0)
        {
        SetMode(3);
        strcpy(message_string,"ERROR Exit 100:  Can't Detect VGA");
        Message_On_Screen(message_string);
        exit(1);
        }
graphics = 1;
Number_Color = 256;
for ( i = 1; i<=4; i++)
        {
        modes[i] = 0;
        }
width = 1024;
height = 768;
mode = Find_Mode(width,height,Number_Color,graphics);
if (mode != -1) modes[4] = mode;
width = 800;
height = 600;
mode = Find_Mode(width,height,Number_Color,graphics);
if (mode != -1) modes[3] = mode;
width = 640;
height = 480;
mode = Find_Mode(width,height,Number_Color,graphics);
if (mode != -1) modes[2] = mode;
width = 320;
height = 200;
mode = Find_Mode(width,height,Number_Color,graphics);
if (mode != -1) modes[1] = mode;
Set_Mode(3);

start:
disp_Init(def_ModeCurrent, NULL);
if (colorflag == 0) frame = frame_Open(main_frame, bd_1, 0x04, 0x47, 0x07);
if (colorflag == 1) frame = frame_Open(main_frame, bd_1, 0x02, 0x7A, 0x07);
if (colorflag == 2) frame = frame_Open(main_frame, bd_1, 0x01, 0x7B, 0x07);
if (colorflag == 3) frame = frame_Open(main_frame, bd_1, 0x07, 0x70, 0x07);
frame_Repaint(frame);
frame_Go(frame, ' ', NULL);
frame_Close(frame);
disp_Close();
goto start;
}
```

Waterways Experiment Station Cataloging-in-Publication Data

Ellis, Michael G.
An image processing technique for achieving lossy compression of data at ratios in excess if 100:1 / by Michael G. Ellis ; prepared for Department of the Army, US Army Corps of Engineers.
217 p. : ill. ; 28 cm. — (Technical report ; ITL-92-10)
Includes bibliographical references.
1. Image processing — Digital techniques. 2. Data compression (Telecommunication) — Computer programs. 3. Digital filters. 4. Optical transfer function. I. United States. Army. Corps of Engineers. II. US Army Engineer Waterways Experiment Station. III. Title. IV. Series: Technical report (US Army Engineer Waterways Experiment Station) ; ITL-92-10.
TA7 W34 no.ITL-92-10